



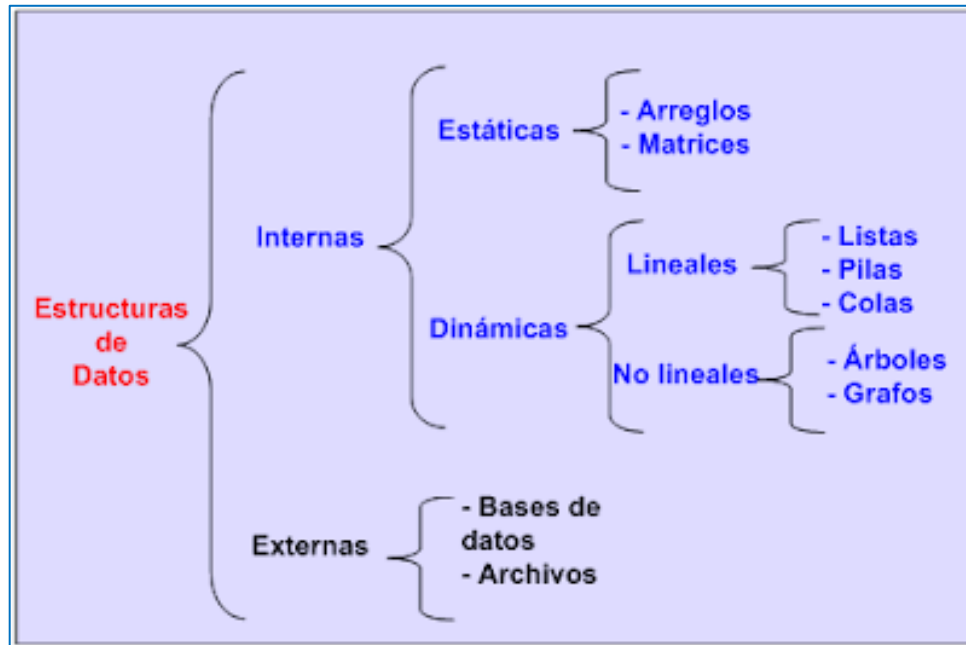
# Taller de Desarrollo de Aplicaciones con Machine Learning

# MATERIAL TÉCNICO DE APOYO

## TAREA N° 01

## Crea programas de Python en Google Colab.

## 1. ESTRUCTURAS DE DATOS, DE CONTROL Y BUCLES CON PYTHON.

Estructura de datos.

- **Estructuras de datos:** Python proporciona varias estructuras de datos incorporadas que son esenciales para organizar y manipular información:

- **Listas (list):** Secuencias ordenadas de elementos que permiten valores duplicados y son modificables.

➤ **Ejemplo:**

```
frutas = ['manzana', 'banana', 'cereza']
```

- **Tuplas (tuple):** Similares a las listas, pero son inmutables (no se pueden cambiar).

➤ **Ejemplo:**

```
colores = ('rojo', 'verde', 'azul')
```

- **Conjuntos (set):** Colecciones no ordenadas de elementos únicos.

➤ **Ejemplo:**

```
numeros = {1, 2, 3, 4, 5}
```

- **Diccionarios (dict):** Pares de clave-valor que permiten una rápida recuperación de valores a través de las claves.

➤ **Ejemplo:**

```
estudiante = {'nombre': 'Juan', 'edad': 21}
```

- **Estructuras de control:** Son utilizadas para controlar el flujo del programa:
  - **Condicionales (if, elif, else):** Permiten la ejecución condicional de código.

```
if Condición :
    #código
    #código
elif Condición :
    #código
    #código
else:
    #código
    #código
#código fuera del if/else
```

Estructura condicional en Python

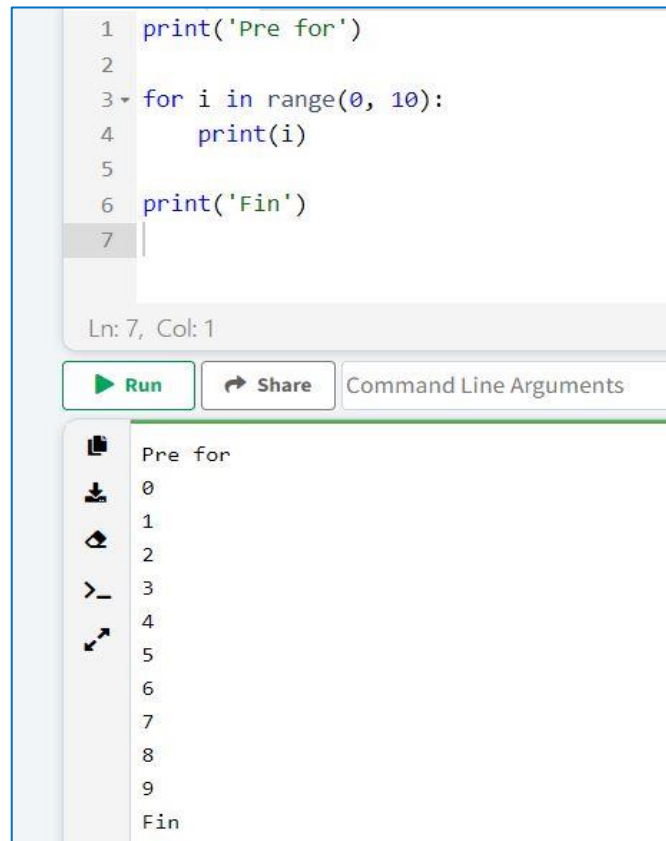
➤ **Ejemplo:**

```
edad = 18
if edad >= 18:
    print("Eres mayor de edad.")
else:
    print("Eres menor de edad.")
```

- **Bucles:** Los bucles permiten repetir bloques de código varias veces:
  - **for:** Se utiliza para iterar sobre secuencias como listas o rangos.

➤ **Ejemplo:**

```
for fruta in frutas:
    print(fruta)
```



```

1 print('Pre for')
2
3 for i in range(0, 10):
4     print(i)
5
6 print('Fin')
7

```

Ln: 7, Col: 1

Run Share Command Line Arguments

Pre for  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
Fin

Ejemplo de ejecución de for en Python

- **while:** Repite una acción mientras una condición sea verdadera.

➤ **Ejemplo:**

```

i = 0
while i < 5:
    print(i)
    i += 1

```

- **Estructuras de datos avanzadas.**

Además de las listas, diccionarios y tuplas, Python ofrece estructuras más avanzadas que son útiles en el desarrollo de aplicaciones complejas, incluyendo Machine Learning:

- **Conjuntos (Sets):** Almacenan elementos únicos y no permiten duplicados. Se utilizan en operaciones matemáticas y para eliminar duplicados de listas.

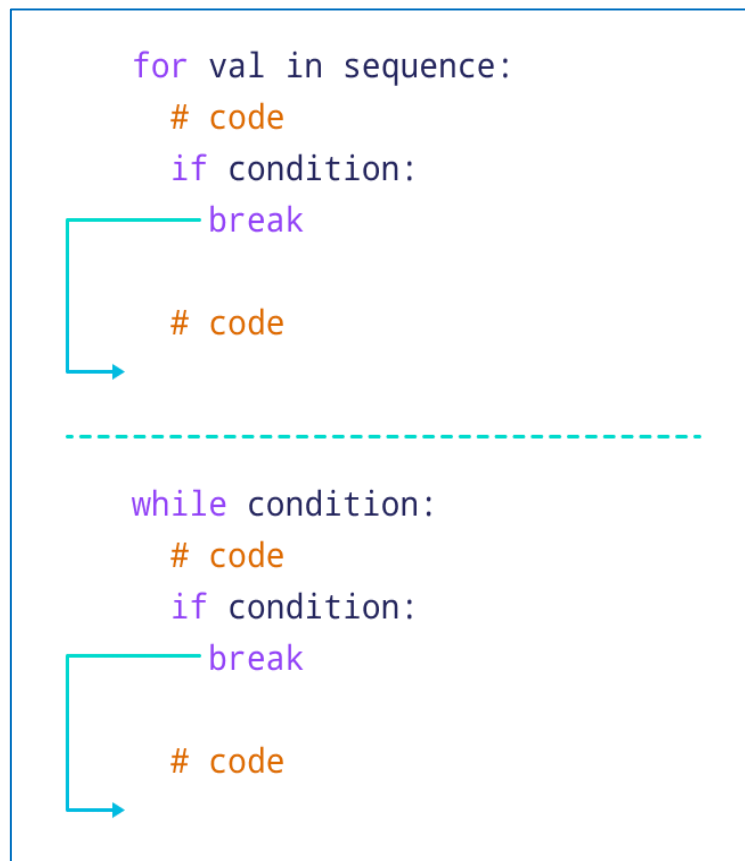
```
conjunto = {1, 2, 3, 3, 4} # El valor 3 se almacena solo una vez
```

- **Listas por comprensión:** Una forma concisa de crear listas nuevas a partir de otras secuencias.

```
cuadrados = [x**2 for x in range(10)]
```

- **Colas y pilas:** Se pueden implementar utilizando listas o la librería `collections.deque` para lograr operaciones más eficientes.
- **Bucles adicionales y manejo de excepciones**
  - **Break y continue en bucles:**
    - **break:** Termina el bucle prematuramente.
    - **continue:** Salta a la siguiente iteración del bucle sin ejecutar el resto del código dentro de la iteración actual.

```
for i in range(10):
    if i == 5:
        break # Se detiene cuando i es igual a 5
```



[Uso de break y continue en Python](#)

- **Manejo de Excepciones:** En Python, las excepciones se pueden manejar utilizando bloques `try-except` para evitar que errores detengan la ejecución del programa.

```
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("Error: No se puede dividir entre cero.")
```

## 2. DEFINICIÓN DE CPUS, GPUS Y TPUS.

- **CPU (Unidad central de procesamiento):** La CPU es el cerebro principal del ordenador, responsable de realizar cálculos, tomar decisiones y ejecutar las instrucciones de los programas. Son excelentes para tareas de cómputo secuencial y se usan en aplicaciones generales.

La CPU es el componente central de cualquier computadora y es responsable de ejecutar instrucciones. Para aplicaciones de Machine Learning, las CPUs son capaces de ejecutar tareas complejas, aunque no están diseñadas específicamente para procesar grandes volúmenes de datos paralelos, como es el caso de los algoritmos de Machine Learning.

- **Ventajas:**

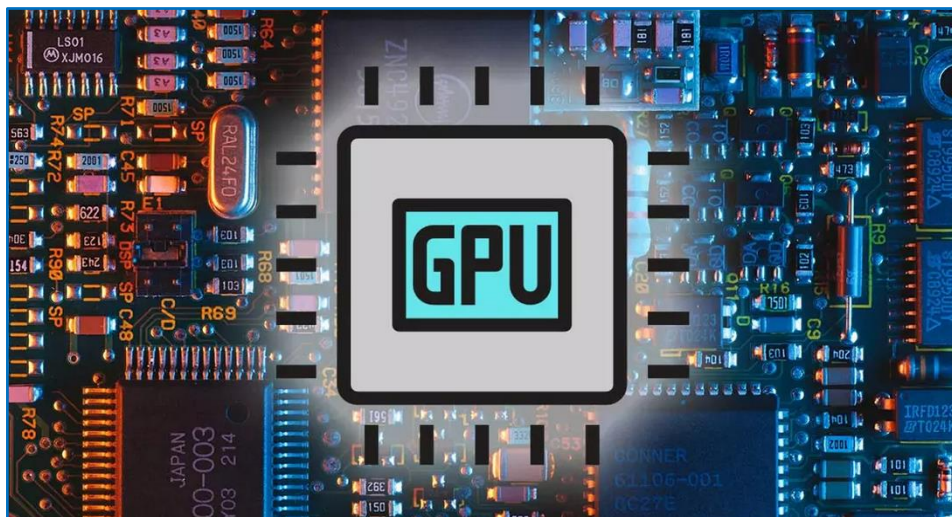
- Buena en tareas generales y operaciones secuenciales.
- Capaz de manejar procesos variados.

- **Limitaciones:**

- No está optimizada para cálculos paralelos masivos, lo que la hace menos eficiente en entrenar grandes modelos de ML comparado con GPUs o TPUs.

- **Ejemplo:** Procesar datos, ejecutar aplicaciones.

- **GPU (Unidad de Procesamiento Gráfico):** Las GPUs son especializadas en procesar gráficos, pero también son útiles para tareas paralelas intensivas como el procesamiento de grandes matrices o los algoritmos de Machine Learning. Son mucho más rápidas que las CPUs en el procesamiento paralelo.



**GPU (Unidad de Procesamiento Gráfico)**

Las GPUs están diseñadas para manejar grandes cantidades de cálculos simultáneos. Originalmente diseñadas para gráficos, las GPUs han demostrado ser muy eficientes en el entrenamiento de modelos de Machine Learning debido a su capacidad de procesamiento en paralelo.

- **Ventajas:**

- Optimizadas para operaciones matriciales, fundamentales en el entrenamiento de redes neuronales.
- Aceleración de cálculos intensivos en álgebra lineal, como los que requieren los modelos de ML.

- **Limitaciones:**

- Más costosas y consumen más energía en comparación con CPUs.

- **Ejemplo:** Entrenamiento de redes neuronales profundas, renderizado de gráficos.

- **TPU (Unidad de procesamiento de tensor):** Son chips diseñados específicamente por Google para acelerar las cargas de trabajo de Machine Learning, en particular el entrenamiento y la inferencia de redes neuronales con TensorFlow.

Las TPUs, desarrolladas por Google, están específicamente diseñadas para ejecutar cargas de trabajo de Machine Learning. Son hardware optimizado para ejecutar modelos de aprendizaje profundo a gran escala de manera eficiente y a alta velocidad.

- **Ventajas:**

- Extremadamente rápidas para ejecutar modelos de TensorFlow.
- Mejor rendimiento energético que las GPUs.
- Diseñadas específicamente para el aprendizaje profundo.

- **Limitaciones:**

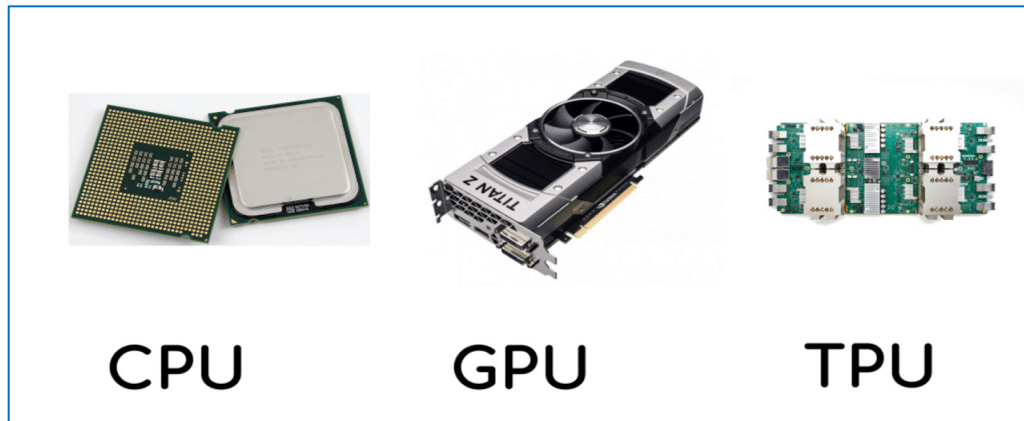
- Menos flexibles que las GPUs para aplicaciones no relacionadas con ML.
- Su uso está mayormente restringido a la infraestructura de Google (Google Cloud).

- **Ejemplo:** TPUs son usadas en Google Colab para acelerar modelos de Machine Learning.



Google Colab te permite elegir entre CPU, GPU o TPU al ejecutar tus programas de Python, lo que te da la flexibilidad de probar diferentes configuraciones de hardware. Para seleccionar, puedes ir a: Entorno de ejecución > Cambiar tipo de entorno de ejecución.

- **Comparación:**



Dispositivos CPU, GPU y TPU

- **CPU:** Versátil, mejor para tareas generales.
- **GPU:** Ideal para operaciones paralelas masivas, útil para entrenar modelos de ML.
- **TPU:** Optimizada para cargas de trabajo específicas de ML, especialmente en TensorFlow.

### 3. PROGRAMACIÓN ORIENTADA A OBJETOS.

- **Conceptos claves de POO en Python:** La Programación Orientada a Objetos (POO) es un paradigma de programación que organiza el código en "objetos", que son instancias de clases. Las clases agrupan datos y comportamientos, lo que facilita la reutilización y la modularidad del código. La Programación Orientada a Objetos es un paradigma que utiliza objetos (instancias de clases) y tiene cuatro principios fundamentales:
  - **Clases y objetos:** Una clase define una plantilla para crear objetos. Cada objeto es una instancia de una clase y contiene atributos (datos) y métodos (funciones).
  - **Clase:** Es un plano o plantilla para crear objetos.

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
```



- **Objeto:** Es una instancia de una clase.

```
persona1 = Persona('Juan', 21)
print(persona1.nombre) # Imprime: Juan
```

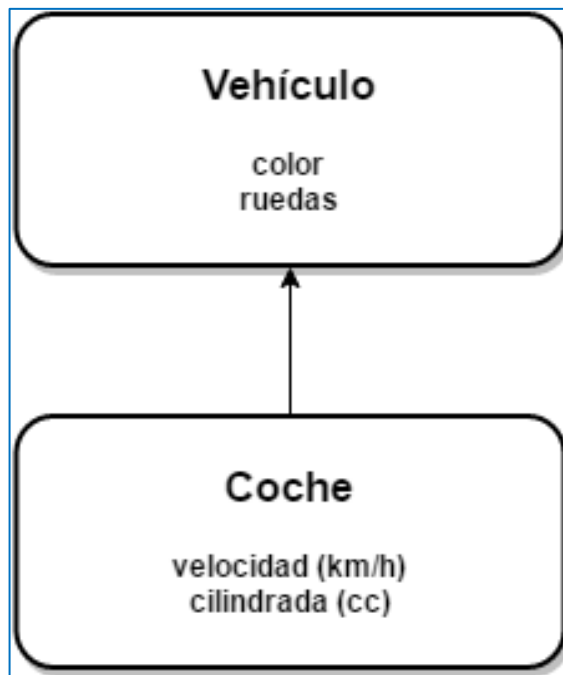
- **Encapsulamiento:** Es el principio de ocultar los detalles internos de una clase y proteger los datos.

- **Ejemplo:** Hacer que ciertos atributos sean privados añadiendo `__` antes del nombre.

```
class Banco:
    def __init__(self, saldo):
        self.__saldo = saldo # Atributo privado
    def mostrar_saldo(self):
        return self.__saldo
```

- **Herencia:** Permite crear nuevas clases que heredan atributos y métodos de una clase existente.

- **Ejemplo:**



[Diagrama de la clase vehículo con la clase coche](#)

```
class Vehiculo():

    def __init__(self, color, ruedas):
        self.color = color
        self.ruedas = ruedas

    def __str__(self):
```

```

        return "Color {}, {} ruedas".format( self.color, self.ruedas )

class Coche(Vehiculo):

    def __init__(self, color, ruedas, velocidad, cilindrada):
        self.color = color
        self.ruedas = ruedas
        self.velocidad = velocidad
        self.cilindrada = cilindrada

    def __str__(self):
        return "color {}, {} km/h, {} ruedas, {} cc".format( self.color,
        self.velocidad, self.ruedas, self.cilindrada )

coche = Coche("azul", 150, 4, 1200)
print(coche)

```

- **Polimorfismo:** Permite que diferentes clases puedan ser tratadas como si fueran de la misma clase a través de una interfaz común.

➤ **Ejemplo:**

```

class Gato(Animal):
    def hacer_sonido(self):
        print(f"{self.nombre} maúlla")

animales = [Perro("Rex"), Gato("Luna")]

for animal in animales:
    animal.hacer_sonido()

```

- **Abstracción:** Proceso de ocultar detalles complejos y mostrar solo lo esencial.

➤ **Ejemplo:**

```

from abc import ABC, abstractmethod

class Figura(ABC):
    @abstractmethod
    def area(self):
        pass

class Circulo(Figura):
    def __init__(self, radio):

```

```

self.radio = radio

def area(self):
    return 3.14 * self.radio ** 2

c = Circulo(5)
print(c.area()) # Imprime el área del círculo

```

- **Ventajas de POO en Machine Learning:**

- Facilita la modularización y reutilización del código.
- Permite modelar entidades del mundo real, como redes neuronales o pipelines de datos.
- Mejora la organización en proyectos de gran escala.

- **Aplicaciones en Machine Learning:**

- La POO es especialmente útil en Machine Learning, donde puedes definir clases para modelos, datasets, o preprocesamiento, facilitando la reutilización y modularidad del código.
- Por ejemplo, podrías crear una clase para definir un ModeloML que encapsule todo el proceso de entrenamiento y evaluación de un modelo.

- **Ejemplo:**

```

from sklearn.linear_model import LinearRegression

class ModeloRegresion:
    def __init__(self):
        self.modelo = LinearRegression()

    def entrenar(self, X, y):
        self.modelo.fit(X, y)

    def predecir(self, X):
        return self.modelo.predict(X)

```

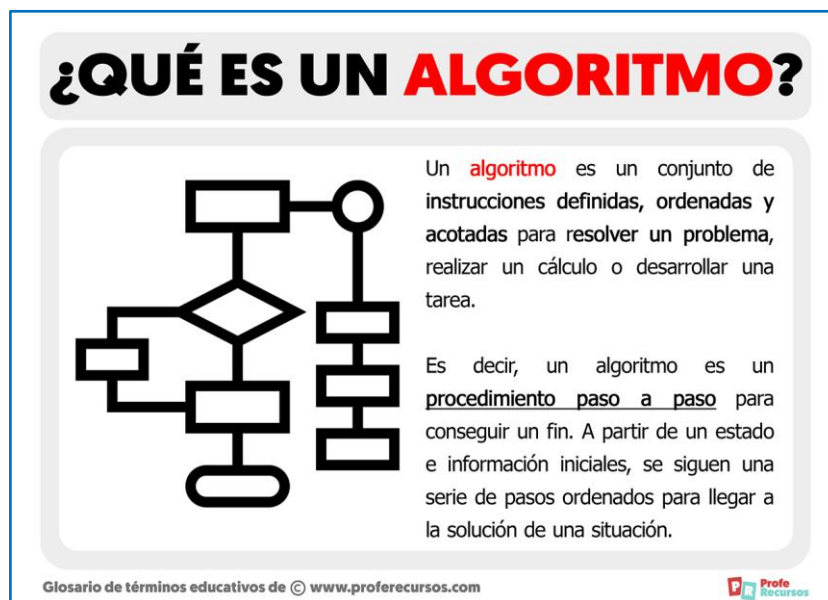
## TAREA N°02

## Crea y entrena modelos ML.

## 1. FUNDAMENTOS DE ALGORITMOS Y PROGRAMACIÓN.

- **Concepto de algoritmo:**

Un algoritmo es una secuencia finita de instrucciones claras y precisas que resuelven un problema o realizan una tarea específica. En el contexto de Machine Learning, los algoritmos son esenciales para entrenar modelos con datos, optimizando parámetros para mejorar la predicción y la clasificación.



[Concepto de algoritmo.](#)

- **Características de un algoritmo:**
  - **Definido:** Cada paso debe estar claramente especificado.
  - **Finito:** Debe terminar en un número limitado de pasos.
  - **Eficiente:** Consume la menor cantidad de recursos posibles (memoria, tiempo).
- **Ejemplos de algoritmos comunes en ML:**
  - **Algoritmo de regresión lineal:** Relaciona una variable dependiente con una o más variables independientes.
  - **Algoritmos de clasificación:** Como K-nearest neighbors (KNN) o Support Vector Machines (SVM).

- **Algoritmo de agrupamiento K-means:** Agrupa puntos de datos en clusters basados en similitudes.

- **Complejidad de un algoritmo:**

La eficiencia de un algoritmo se evalúa utilizando la notación Big O, que describe el peor caso en términos de tiempo o espacio. Esto es clave en ML, ya que ciertos algoritmos pueden volverse ineficientes para grandes volúmenes de datos.

Python es uno de los lenguajes de programación más usados en Machine Learning debido a su simplicidad, legibilidad y la gran cantidad de bibliotecas especializadas.

## 2. ESTRUCTURAS DE DATOS, DE CONTROL Y BUCLES CON PYTHON.

En Python, hay varias estructuras de datos (listas, diccionarios, tuplas, conjuntos) que son esenciales para organizar la información. También existen estructuras de control que permiten la ejecución de código condicional y la repetición de acciones:

- **Condicionales (if, else, elif):** Permiten la toma de decisiones.
- **Bucles (for, while):** Facilitan la repetición de tareas de forma eficiente.

Ejemplo de un bucle en Python:

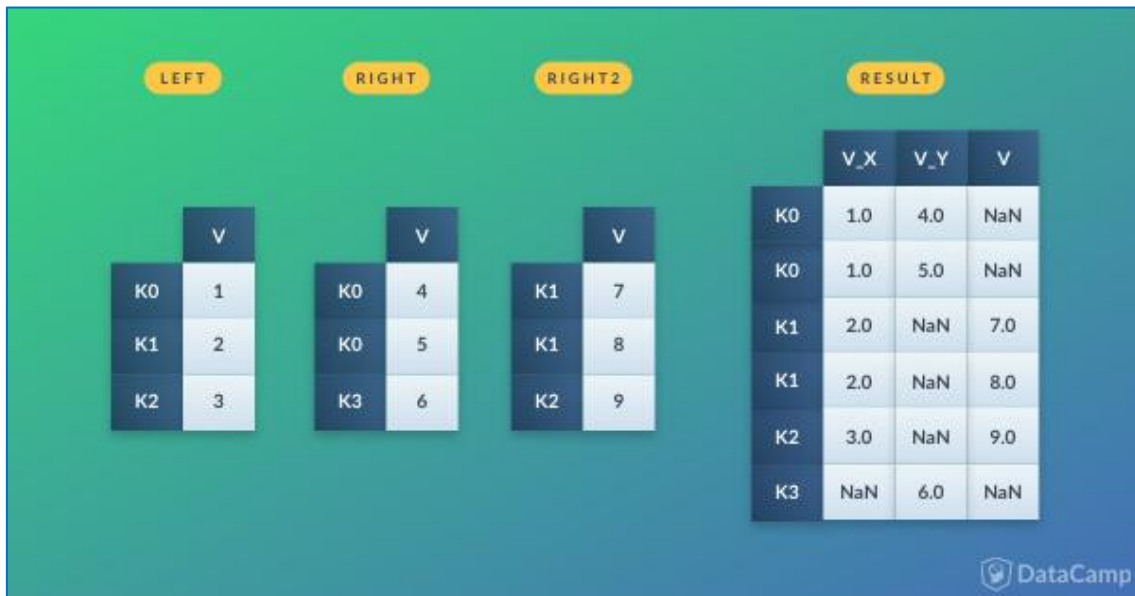
```
for i in range(5):
    print(i) # Imprime números del 0 al 4
```

- **Estructuras de datos adicionales:**

Además de las listas, tuplas, y diccionarios que ya mencionamos, es importante conocer estructuras más avanzadas para mejorar el rendimiento y la organización de los datos en Machine Learning:

- **Pandas DataFrames:** Son fundamentales para la manipulación de grandes conjuntos de datos. Facilitan la organización y análisis de datos tabulares.

```
import pandas as pd
datos = {'Edad': [22, 23, 24], 'Nombre': ['Ana', 'Luis', 'Carlos']}
df = pd.DataFrame(datos)
```



[Representación gráfica de pandas dataframes](#)

- **NumPy Arrays:** Permiten realizar operaciones matemáticas de manera eficiente, gracias a su capacidad de manipular matrices y vectores de manera optimizada.

```
import numpy as np
matriz = np.array([[1, 2, 3], [4, 5, 6]])
```

### 3. CONTROL DE VERSIONES DE SOFTWARE GITHUB.

GitHub es una plataforma de control de versiones y colaboración, utilizada ampliamente para el desarrollo de software. Git es la herramienta que permite rastrear cambios en el código, permitiendo trabajar en equipo sin conflictos.

- **Comandos básicos en Git:**

- **git init:** Inicializa un repositorio local.
- **git add:** Añade archivos al área de preparación.
- **git commit -m "mensaje":** Guarda los cambios en el repositorio.
- **git push origin master:** Sube los cambios al repositorio en GitHub.

- **Ejemplo en Google Colab para clonar un repositorio:**

```
!git clone https://github.com/usuario/repositorio.git
```

- **Branches y Merge:**

Para equipos que desarrollan modelos de Machine Learning colaborativamente, es fundamental trabajar en ramas (branches) para

evitar conflictos de código. Git permite crear una nueva rama para desarrollar una nueva característica, entrenar un modelo distinto, o realizar ajustes experimentales sin afectar la rama principal.

- **Branching:**

```
git branch nueva_rama  
git checkout nueva_rama
```

- **Merge:** Después de probar los cambios en la nueva rama, se puede integrar a la rama principal utilizando el comando merge.

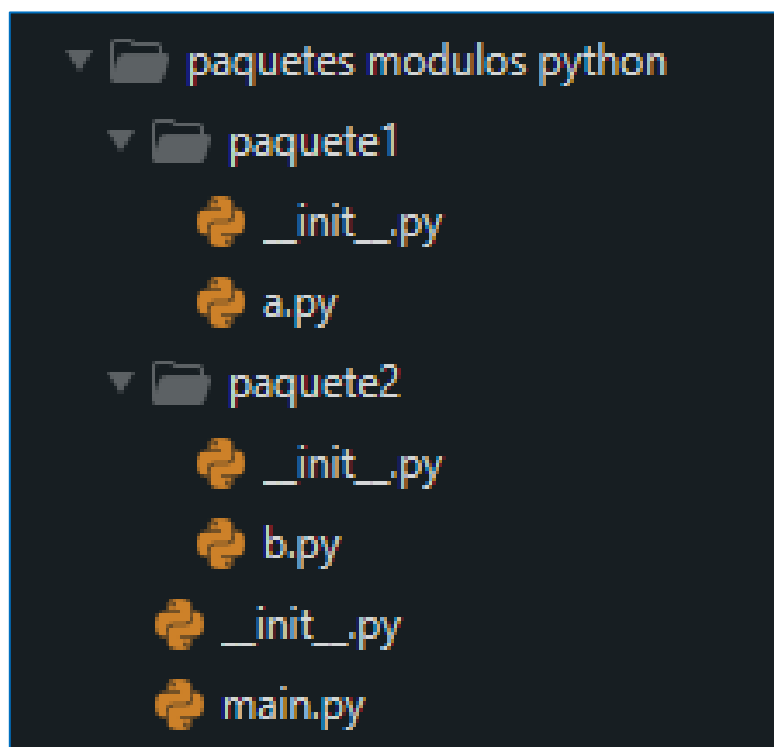
```
git checkout main  
git merge nueva_rama
```

- **Commits y pull requests:**

Hacer commits frecuentes con descripciones claras permite un mejor control de los cambios realizados. En proyectos colaborativos, es común realizar pull requests para que los cambios sean revisados por otros miembros del equipo antes de ser integrados en la rama principal.

#### 4. INSTALAR E IMPORTAR PAQUETES EN PYTHON.

En Python, las bibliotecas o paquetes son colecciones de módulos que extienden la funcionalidad de base del lenguaje. Puedes instalar y gestionar paquetes a través de pip.



Ejemplo de organización de paquetes



- **Para instalar un paquete en Google Colab:**

```
!pip install nombre_paquete
```

- **Para importar un paquete en Python:**

```
import nombre_paquete
```

- **Ejemplo en Python:**

```
!pip install numpy
import numpy as np
```

- **Pip y entornos virtuales:**

Una herramienta común para instalar paquetes en Python es pip. Sin embargo, para evitar conflictos entre diferentes versiones de paquetes en distintos proyectos, es recomendable utilizar entornos virtuales que permiten tener un conjunto aislado de dependencias para cada proyecto.

- Crear un entorno virtual:

```
python -m venv mi_entorno
```

- Activar el entorno:

- En Windows:

```
mi_entorno\Scripts\activate
```

- En Linux/Mac:

```
source mi_entorno/bin/activate
```

- Crear un entorno virtual:

```
python -m venv mi_entorno
```

- **Instalar un paquete usando pip:**

```
pip install nombre_paquete
```

## 5. LIBRERIAS DE MACHINE LEARNING.

En Python, existen varias librerías especializadas en Machine Learning que facilitan el desarrollo y entrenamiento de modelos:

- **Tensorflow.**

TensorFlow es una plataforma de código abierto creada por Google que permite construir, entrenar y desplegar modelos de Machine Learning. Se utiliza principalmente para redes neuronales y deep learning.



[Logo de TensorFlow](#)

Es una de las librerías más potentes y populares para crear y entrenar modelos de Machine Learning. Está diseñada para trabajar con redes neuronales y se utiliza tanto en investigaciones como en aplicaciones industriales. En TensorFlow, los modelos se definen usando grafos de datos, lo que permite la optimización y el entrenamiento en CPUs, GPUs y TPUs.

- Flujo básico de TensorFlow:

```
import tensorflow as tf
# Definir una capa densa
capa = tf.keras.layers.Dense(units=1, input_shape=[1])
```

- Para instalar TensorFlow en Google Colab:

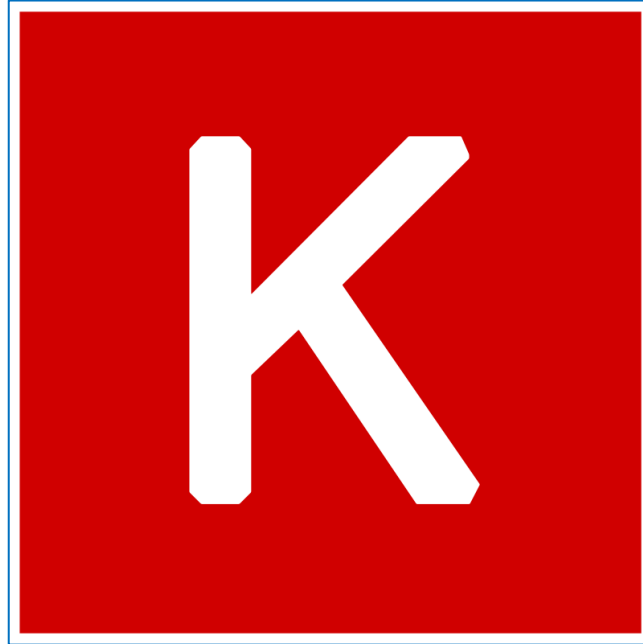
```
!pip install tensorflow
```

- Ejemplo básico en TensorFlow:

```
import tensorflow as tf
print(tf.__version__)
```

- **Keras.**

Es una interfaz de alto nivel para construir modelos de redes neuronales en TensorFlow. Simplifica la creación de modelos, abstractando muchos de los detalles complejos de TensorFlow.



Logo de Keras.

Keras es una API de alto nivel construida sobre TensorFlow, diseñada para facilitar la creación rápida de redes neuronales. Es muy popular por su simplicidad y facilidad de uso.

- Ejemplo de un modelo secuencial con Keras:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

modelo = Sequential([
    Dense(128, activation='relu', input_shape=(100,)),
    Dense(1, activation='linear')
])
```

- Ejemplo en Keras para construir una red neuronal:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Crear un modelo secuencial
modelo = Sequential()
modelo.add(Dense(10, input_shape=(2,), activation='relu')) # Capa
oculta
modelo.add(Dense(1, activation='sigmoid')) # Capa de salida
```

- **Matplotlib.**

Es una librería esencial para la visualización de datos en Python. En Machine Learning, es utilizada para graficar datos y resultados de modelos, como curvas de pérdida o precisión.



[Logo de matplotlib](#)

Es una biblioteca para crear gráficos y visualizaciones, ideal para explorar y presentar datos.

- Ejemplo básico:

```
import matplotlib.pyplot as plt

# Crear un gráfico simple
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]
plt.plot(x, y)
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('Gráfico de ejemplo')
plt.show()
```

## 6. FUNDAMENTOS DE REDES NEURONALES ARTIFICIALES.

Las Redes Neuronales Artificiales (ANN) son modelos de aprendizaje inspirados en el cerebro humano. Están compuestas por:

- **Neuronas:** Las neuronas son unidades que reciben varias entradas (pesadas por parámetros), realizan una operación matemática y entregan una salida.
- **Capas:** Una capa es un conjunto de neuronas que procesan entradas y pasan su salida a la siguiente capa.
- **Pesos y sesgos:** Los pesos determinan la importancia de cada entrada, mientras que el sesgo ayuda a ajustar las salidas de la red.
- **Funciones de activación:** Son funciones no lineales que permiten a la red aprender relaciones complejas. Ejemplos comunes son ReLU y sigmoide.

Las redes neuronales aprenden ajustando los pesos a través de un proceso de entrenamiento, utilizando algoritmos como el Gradiente Descendente y una función de pérdida que mide el error.

- **Proceso de entrenamiento:**

El entrenamiento de una red neuronal implica ajustar los pesos y sesgos a través de la retropropagación y el uso de un algoritmo de optimización como Adam o SGD (descenso de gradiente estocástico).

- **Ejemplo básico de una ANN con Keras:**

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Crear una red neuronal simple
modelo = Sequential()
modelo.add(Dense(128, input_dim=784, activation='relu')) # Capa oculta
modelo.add(Dense(10, activation='softmax')) # Capa de salida

# Compilar el modelo
modelo.compile(optimizer='adam', loss='categorical_crossentropy',
               metrics=['accuracy'])
```

## 7. FUNDAMENTOS DE REGRESIÓN LINEAL.

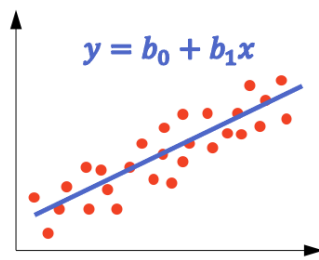
- **Concepto:**

La regresión lineal es un método estadístico utilizado para modelar la relación entre una variable dependiente y una o más variables independientes. En Machine Learning, es el algoritmo más sencillo para predecir valores numéricos continuos.

La Regresión Lineal es un modelo de aprendizaje supervisado utilizado para predecir un valor continuo en base a una variable independiente. La relación entre las variables se representa mediante una línea recta:

- **Ecuación de la recta:**  $y = mx + b$ , donde  $m$  es la pendiente y  $b$  es la intersección con el eje Y.

El objetivo de la regresión lineal es minimizar la función de costo (error), generalmente mediante el Error Cuadrático Medio (MSE).

**Regresión Lineal Simple**

$$b_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

$$b_0 = \bar{y} - b_1 \bar{x}$$

Fórmula de regresión simple.

- **Entrenamiento del modelo:**

Durante el entrenamiento, el modelo ajusta los valores de los pesos y el sesgo para minimizar la función de pérdida, como el error cuadrático medio (MSE).

- Ejemplo en Python utilizando scikit-learn:

```
from sklearn.linear_model import LinearRegression
import numpy as np
```

```
# Datos de entrenamiento
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([5, 7, 9, 11, 13])
```

```
# Crear el modelo de regresión lineal
modelo = LinearRegression()
modelo.fit(X, y)
```

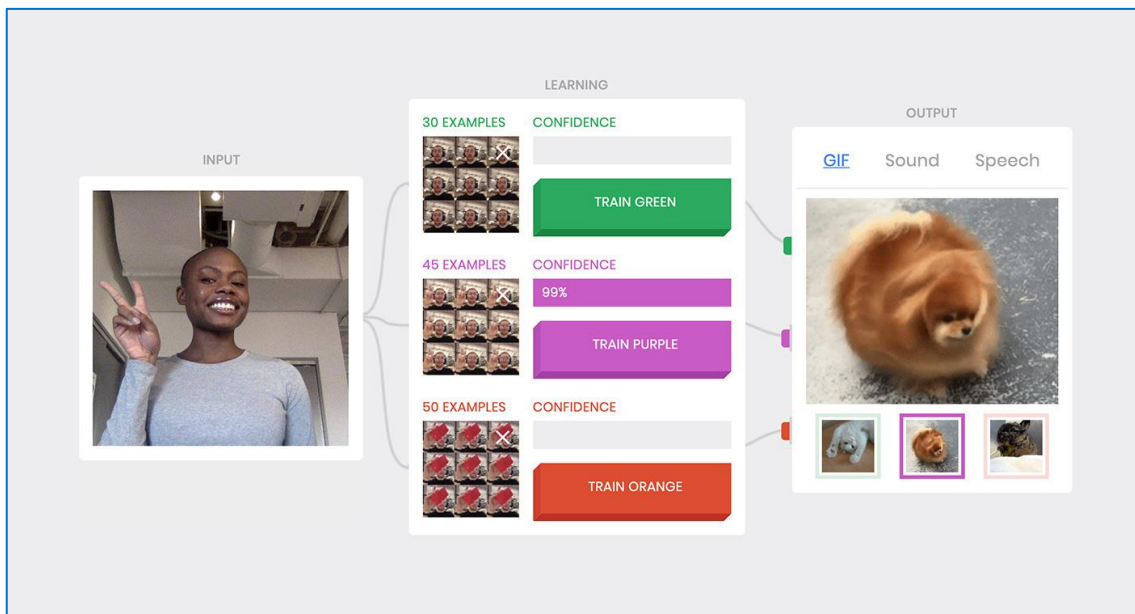
```
# Realizar una predicción
prediccion = modelo.predict([[6]])
print(f"Predicción para X=6: {prediccion}")
```

## TAREA N° 03

## Exporta e integra modelos de Machine Learning.

### 1. ENTORNO DEL SERVICIO WEB TEACHABLE MACHINE.

Teachable Machine es una plataforma creada por Google que permite a los usuarios crear modelos de Machine Learning de manera fácil y rápida, sin necesidad de tener experiencia técnica en programación. Con Teachable Machine, puedes entrenar modelos directamente en el navegador utilizando datos de imágenes, sonidos o posturas.



[Vista de la plataforma Teachable Machine](#)

- **Características de Teachable Machine:**

Teachable Machine es una herramienta desarrollada por Google que permite a los usuarios crear modelos de ML de manera sencilla y accesible, sin necesidad de tener experiencia previa en programación. Sus características clave incluyen:

- **Interfaz visual intuitiva:** Los usuarios pueden cargar datos (imágenes, sonidos o poses) directamente a través de la interfaz gráfica sin necesidad de escribir código.
- **Entrenamiento rápido:** Teachable Machine utiliza algoritmos preentrenados y optimizados, lo que permite a los usuarios ver los resultados de sus modelos en minutos.
- **Exportación fácil:** Los modelos entrenados se pueden exportar fácilmente a TensorFlow.js, TensorFlow o como archivos de imagen o audio.



- **Proceso de trabajo:**

- **Entrenar un modelo:** Subes imágenes, grabas audio o muestras datos de posturas para que el modelo aprenda.
- **Exportar el modelo:** Una vez entrenado, puedes descargar el modelo en un formato compatible con TensorFlow, TensorFlow.js o TensorFlow Lite.

- **Aplicaciones prácticas:**

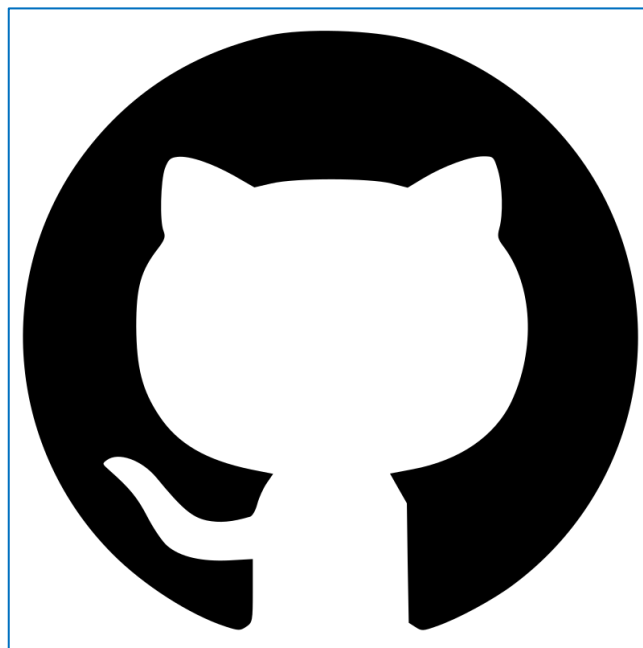
- **Prototipos rápidos:** Ideal para educadores y desarrolladores que deseen crear prototipos de aplicaciones de reconocimiento de imágenes o sonidos sin profundizar en la codificación.
- **Integración en proyectos:** Los modelos generados pueden integrarse en aplicaciones web y móviles, lo que permite su uso en diversos contextos.

- **Ejemplo de uso de Teachable Machine:**

- Entrenas un modelo para reconocer gestos de la mano y luego lo exportas para su uso en una aplicación web utilizando TensorFlow.js.

## 2. CONTROL DE VERSIONES DE SOFTWARE CON GITHUB.

- **Concepto**



[Logo de Github.](#)

GitHub es una plataforma popular para el control de versiones, que permite colaborar y gestionar cambios en proyectos de software.

Utilizando Git, puedes:

- Rastrear los cambios realizados en tu código.
- Colaborar en equipo en el desarrollo de modelos de Machine Learning.
- Desplegar aplicaciones web que integran modelos de ML.

- **Flujo básico con GitHub:**

- **Clonar un repositorio:** Para obtener una copia local de un proyecto:

```
git clone https://github.com/usuario/proyecto.git
```

- **Subir cambios:** Después de hacer modificaciones, los pasos son:

```
git add .  
git commit -m "Mensaje sobre los cambios"  
git push origin master
```

- **Configuración del repositorio:**

Crear un repositorio en GitHub es el primer paso para el control de versiones. Los pasos básicos incluyen:

- Crear un nuevo repositorio:
  - Ingresar a GitHub y seleccionar "New Repository".
  - Elegir un nombre descriptivo y, opcionalmente, añadir una descripción.
- Clonar el repositorio:
  - Para trabajar localmente, los usuarios deben clonar el repositorio a su máquina:

```
git clone https://github.com/usuario/nombre_repositorio.git
```

- **Colaboración en proyectos:**

- Forks y Pull Requests: Los usuarios pueden crear forks de repositorios para realizar cambios y, posteriormente, enviar un pull request para que los propietarios del repositorio original revisen e integren sus contribuciones.
- Issues: GitHub permite crear issues para discutir errores, solicitar nuevas características o hacer seguimiento de tareas, lo que mejora la colaboración en proyectos.

### 3. EXPORTAR/GUARDAR EL MODELO ENTRENADO KERAS HDF5 (NOMBRE\_ARCHIVO.H5) CON PYTHON.

Después de entrenar un modelo de Machine Learning utilizando Keras (una API de alto nivel de TensorFlow), puedes exportarlo para su reutilización o implementación en otras plataformas.

- **Pasos para guardar el modelo en formato HDF5 (.h5):**

```
# Guardar el modelo en un archivo .h5
modelo.save('nombre_modelo.h5')
```

Esto genera un archivo en formato HDF5, que contiene la estructura del modelo, los pesos y la configuración de entrenamiento. Este archivo puede ser cargado más tarde para continuar con el entrenamiento o para hacer predicciones en otro entorno.

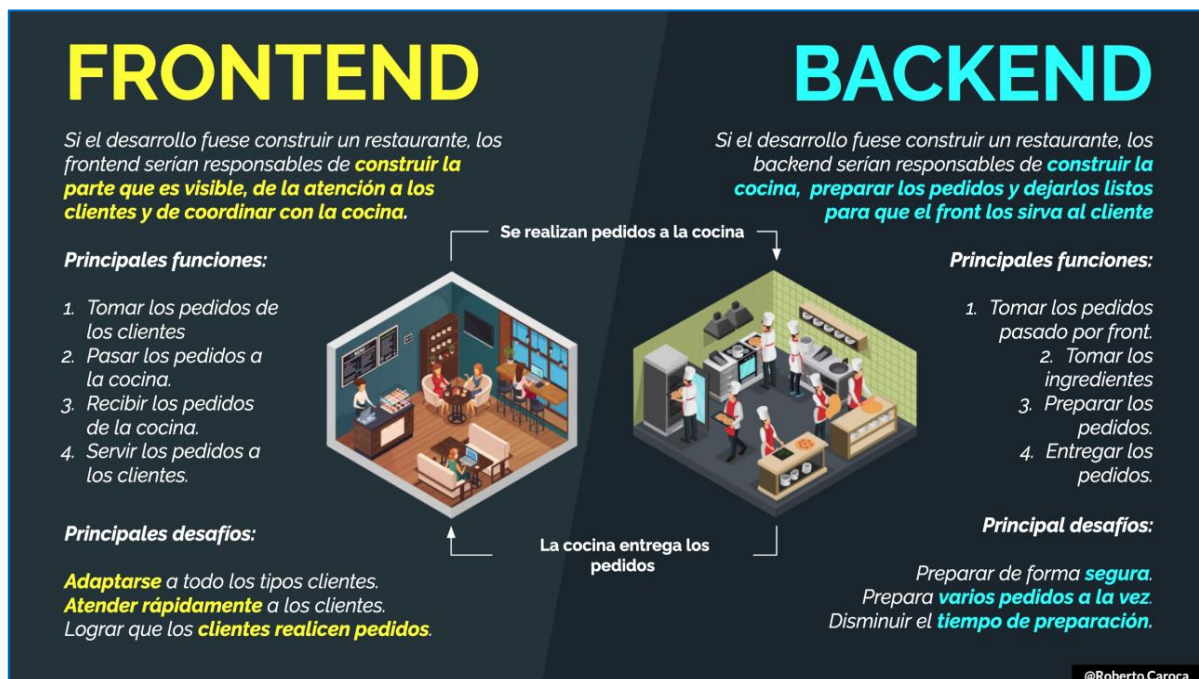
- **Cargar el modelo guardado:**

```
from tensorflow.keras.models import load_model

# Cargar el modelo desde el archivo .h5
modelo_cargado = load_model('nombre_modelo.h5')
```

Este formato es ideal para trasladar modelos entre plataformas o integrarlos en entornos web y aplicaciones móviles.

### 4. PROGRAMACIÓN CON TECNOLOGÍAS WEB.



Diferencias entre frontend y backend.

- **Frontend.**

El frontend se refiere a la parte de una aplicación web que interactúa directamente con los usuarios, es decir, la interfaz de usuario. Las principales tecnologías utilizadas en el frontend son:

- **HTML:** Estructura la página web.
- **CSS:** Define el estilo y la apariencia de la página web.
- **JavaScript:** Proporciona interactividad y funcionalidad dinámica.

Ejemplo básico de HTML + JavaScript para cargar un modelo entrenado en el navegador:

```
<!DOCTYPE html>
<html>
<head>
  <title>Modelo de ML en el navegador</title>
</head>
<body>
  <h1>Clasificación de imágenes con ML</h1>
  <input type="file" id="imageUpload" />
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
  <script>
    // Aquí se cargará el modelo usando TensorFlow.js
  </script>
</body>
</html>
```

- **Backend.**

El backend se refiere a la parte del servidor de una aplicación web que maneja la lógica de negocio, el almacenamiento de datos y la comunicación con las bases de datos. Las tecnologías comunes en el backend incluyen:

- **Node.js:** Un entorno para ejecutar JavaScript en el servidor.
- **Python:** Usado frecuentemente con frameworks como Flask o Django para crear aplicaciones web que interactúan con modelos de ML.

Ejemplo de un backend simple en Flask que integra un modelo de Machine Learning:

```
from flask import Flask, request, jsonify
from tensorflow.keras.models import load_model
import numpy as np
```

```

app = Flask(__name__)
model = load_model('nombre_modelo.h5')

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json['data']
    prediction = model.predict(np.array([data]))
    return jsonify({'prediction': prediction.tolist()})

if __name__ == '__main__':
    app.run(debug=True)

```

## 5. LIBRERÍA DE MACHINE LEARNING PARA WEB.

- **TensorflowJS.**

TensorFlow.js es una versión de TensorFlow que permite la ejecución de modelos de Machine Learning directamente en el navegador o en Node.js utilizando JavaScript. Es muy útil para aplicaciones web que requieren modelos de ML, ya que no es necesario hacer solicitudes a un servidor externo para ejecutar predicciones.

- **Ventajas:**

- **Ejecuta modelos directamente en el cliente:** Lo que reduce la latencia y permite que los modelos funcionen sin conexión.
- **Facilidad de uso:** Puedes convertir un modelo entrenado en TensorFlow a TensorFlow.js para su uso en el navegador.

## 6. IMPORTA EL MODELO USANDO TENSORFLOWJS Y JAVASCRIPT.



[Logo de Javascript.](#)

Para importar un modelo previamente entrenado con Keras y exportado a

TensorFlow.js, se deben seguir estos pasos:

- **Convertir el modelo Keras a TensorFlow.js:** Utiliza la herramienta tensorflowjs\_converter para convertir el modelo .h5 a un formato compatible con TensorFlow.js.

```
tensorflowjs_converter --input_format keras nombre_modelo.h5
/ruta/modelo_js/
```

- **Cargar el modelo en una aplicación web con TensorFlow.js:** Una vez convertido el modelo, puedes cargarlo en tu página web utilizando JavaScript:

```
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
<script>
  // Cargar el modelo
  async function cargarModelo() {
    const modelo = await
    tf.loadLayersModel('/ruta/modelo_js/model.json');
    console.log('Modelo cargado');
  }
  cargarModelo();
</script>
```

- **Usar el modelo para predicciones:** Una vez que el modelo está cargado, puedes usarlo para hacer predicciones directamente en el navegador:

```
<script>
  async function predecir(datos) {
    const prediccion = modelo.predict(tf.tensor([datos]));
    console.log(prediccion.arraySync());
  }
</script>
```

**TAREA N° 04****Usa herramientas de IA para integrarlo al desarrollo de software.****1. ESTRUCTURAS DE DATOS, DE CONTROL Y BUCLES CON PYTHON.**

Python es uno de los lenguajes más populares en el desarrollo de aplicaciones de Machine Learning y AI debido a su simplicidad y extensa biblioteca. Dentro de este contexto, las estructuras de datos y los bucles son fundamentales para el manejo eficiente de la información.

- **Estructuras de datos.**

- **Listas:** Sirven para almacenar una colección de datos.

```
lista = [1, 2, 3, 4, 5]
```

- **Diccionarios:** Estructuras que almacenan pares clave-valor.

```
diccionario = {'nombre': 'Ana', 'edad': 25}
```

- **Tuplas:** Son listas inmutables.

```
tupla = (10, 20, 30)
```

- **Estructuras de control y bucles.**

- **Condicionales:**

```
if x > 10:
    print("Es mayor a 10")
elif x == 10:
    print("Es igual a 10")
else:
    print("Es menor a 10")
```

- **Bucles:**

- **For loop:** Para iterar sobre una secuencia de datos.

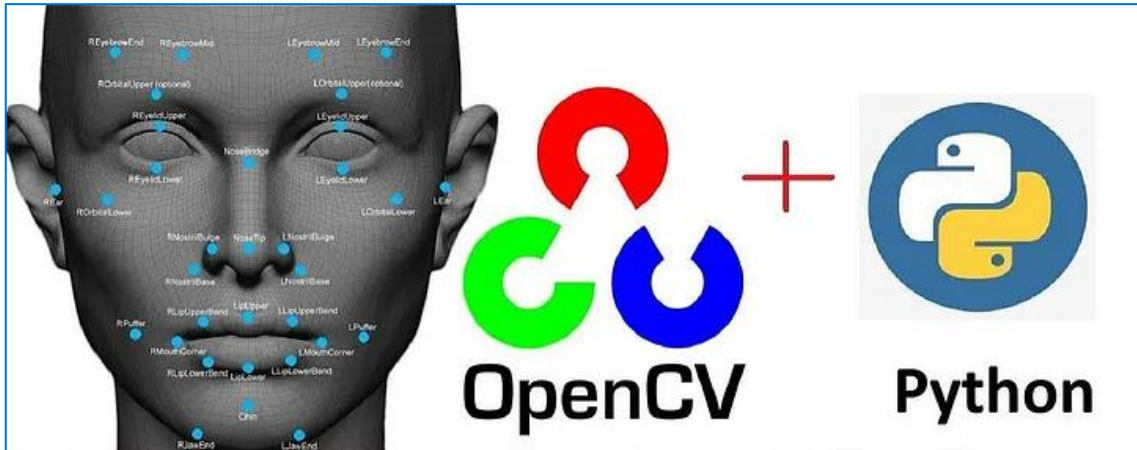
```
for i in range(5):
    print(i)
```



- **While loop:** Se ejecuta mientras la condición sea verdadera.

```
while x < 10:
    x += 1
```

## 2. FUNDAMENTOS DE RECONOCIMIENTO DE OBJETOS CON PYTHON Y OPENCV.



[Python y Opencv](#)

OpenCV (Open Source Computer Vision Library) es una biblioteca poderosa para el procesamiento de imágenes y el reconocimiento de objetos. Usando Python y OpenCV, puedes construir modelos de visión por computadora que identifican y clasifican objetos en imágenes o videos.

- **Ejemplo básico de reconocimiento de objetos:**

- **Instalación de OpenCV:**

```
pip install opencv-python
```

- **Detección de objetos:** El siguiente ejemplo muestra cómo cargar una imagen y detectar un objeto utilizando el clasificador de Haar para reconocer rostros:

```
import cv2
```

```
# Cargar una imagen
img = cv2.imread('imagen.jpg')
```

```
# Cargar el clasificador preentrenado de Haar
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

```
# Convertir la imagen a escala de grises
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
# Detectar rostros
rostros = face_cascade.detectMultiScale(gray, 1.1, 4)

# Dibujar rectángulos alrededor de los rostros
for (x, y, w, h) in rostros:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)

# Mostrar la imagen
cv2.imshow('Rostros detectados', img)
cv2.waitKey(0)
```

Este es un ejemplo básico de cómo se puede usar OpenCV para el reconocimiento de objetos, un paso importante en aplicaciones de visión por computadora.

- **Detección de bordes:**

La detección de bordes es un paso importante en el procesamiento de imágenes que permite encontrar los contornos de los objetos dentro de una imagen, utilizando algoritmos como Canny Edge Detection.

- **Detección de bordes con Canny:**

```
import cv2
imagen = cv2.imread('imagen.jpg', 0)
bordes = cv2.Canny(imagen, 100, 200)
cv2.imshow('Bordes', bordes)
cv2.waitKey(0)
```

- **Detección de rostros:**

OpenCV incluye un clasificador Haar Cascade para la detección de rostros. Esto permite identificar rostros en imágenes y videos.

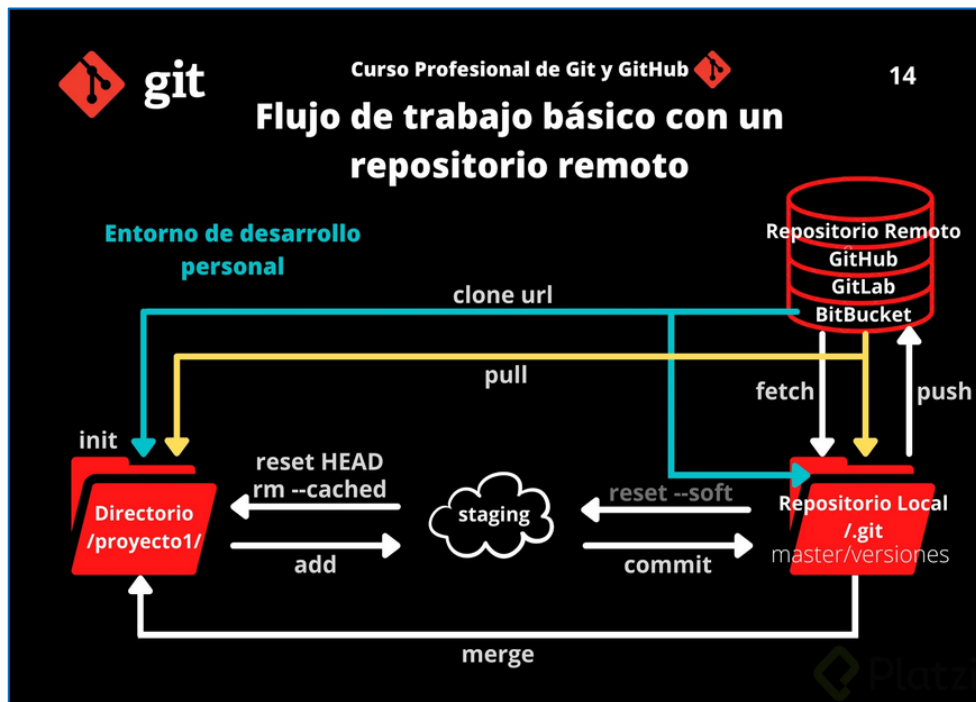
- **Ejemplo de detección de rostros:**

```
rostro_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
rostros = rostro_cascade.detectMultiScale(imagen, 1.1, 4)
for (x, y, w, h) in rostros:
    cv2.rectangle(imagen, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

### 3. CONTROL DE VERSIONES DE SOFTWARE CON GITHUB.

GitHub es una herramienta esencial para el desarrollo colaborativo de software, permitiendo a los equipos gestionar y rastrear los cambios en sus proyectos de manera eficiente.

GitHub utiliza Git, un sistema de control de versiones distribuido, para facilitar el trabajo en proyectos complejos.



[Flujo de trabajo básico con un repositorio remoto](#)

- **Flujo básico con GitHub:**

- Crear un repositorio:
  - Ve a GitHub, crea un nuevo repositorio y clona el proyecto localmente:

```
git clone https://github.com/tu_usuario/proyecto.git
```

- Subir cambios:
  - Añade los cambios al repositorio:

```
git add .
git commit -m "Añadido reconocimiento de objetos con OpenCV"
git push origin master
```

- **Ventajas para Machine Learning:**

- Permite un control detallado de los cambios en los modelos de IA.
- Facilita la colaboración en el desarrollo de modelos, tanto en la creación como en la optimización.

#### 4. PROGRAMACIÓN CON ARDUINO:

Arduino es una plataforma de hardware libre que permite controlar

componentes electrónicos como sensores y actuadores. Es popular para proyectos de IoT y ML embebido.

- **Tipos de datos, condicionales, bucles, funciones, lectura de señales digitales.**

- **Tipos de Datos:**

Los principales tipos de datos en Arduino incluyen int (enteros), float (decimales), char (caracteres), y boolean (verdadero o falso).

- **Condicionales:**

```
if (sensorValue > threshold) {
  // Ejecutar acción
}
```

- **Bucles:**

➤ **Bucle for:** Utilizado para iterar un número determinado de veces.

```
for (int i = 0; i < 10; i++) {
  // Código que se ejecuta 10 veces
}
```

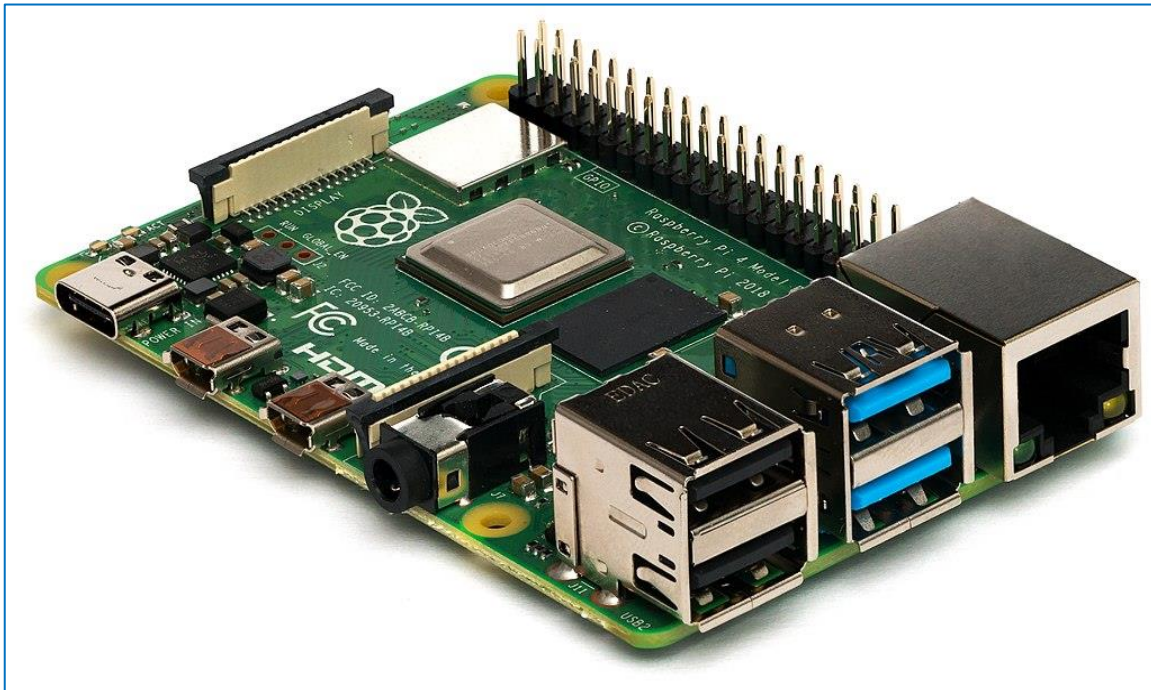
- **Lectura de Señales Digitales:** Los pines digitales de Arduino pueden leer señales de sensores:

```
int valor = digitalRead(2); // Lee el valor del pin digital 2
```

- **Uso de Arduino en AI:** Arduino se puede utilizar para recolectar datos desde el entorno, como datos de temperatura o luz, que luego pueden ser procesados por un modelo de ML embebido.

## 5. INSTALACIÓN DE SISTEMA OPERATIVO PARA RASPBERRY PI.

Raspberry Pi es un ordenador de placa reducida que es ideal para implementar proyectos de IoT y ML embebido.



[Raspberry Pi](#)

La instalación de su sistema operativo (OS) es el primer paso para comenzar a programar.

- **Pasos para instalar el OS:**

- Descargar Raspberry Pi OS desde el sitio oficial.
- Grabar la imagen del sistema operativo en una tarjeta SD utilizando una herramienta como balenaEtcher.
- Insertar la tarjeta SD en la Raspberry Pi y encender el dispositivo.
- Conectar Raspberry Pi a la red y configurar el entorno inicial.

Una vez configurado, Raspberry Pi es capaz de ejecutar aplicaciones de ML de manera eficiente, especialmente cuando se combina con TensorFlow Lite.

## 6. DIFERENCIAS ENTRE TENSORFLOW Y TENSORFLOW LITE.

TensorFlow y TensorFlow Lite son frameworks para el desarrollo de aplicaciones de Machine Learning, pero están diseñados para diferentes propósitos:

- **TensorFlow:** Se utiliza principalmente en entornos de desarrollo y producción a gran escala. Es adecuado para modelos complejos y requiere una considerable cantidad de potencia de procesamiento.
- **TensorFlow Lite:** Es una versión optimizada de TensorFlow diseñada para dispositivos móviles y embebidos. TensorFlow Lite permite ejecutar

modelos de ML en dispositivos de baja potencia, como teléfonos inteligentes y Raspberry Pi.

Diferencias clave:

- **Tamaño del modelo:** TensorFlow Lite genera modelos más pequeños y ligeros.
- **Rendimiento:** TensorFlow Lite está optimizado para ejecución en dispositivos con recursos limitados.
- **Uso:** TensorFlow Lite se usa en aplicaciones móviles, IoT y otros dispositivos embebidos.

## 7. DEFINICIÓN DE TINYML Y COMPUTACIÓN EMBEBIDA.

- **TinyML:**

TinyML es una rama emergente de Machine Learning que se enfoca en la implementación de modelos de ML en dispositivos muy pequeños y de baja potencia, como microcontroladores. Estos modelos son extremadamente eficientes y pueden ejecutarse en hardware limitado sin necesidad de conexión a la nube.

- **Ejemplo de uso:**

Un sensor de movimiento conectado a un microcontrolador Arduino podría usar un modelo TinyML para detectar patrones de movimiento y activar una alarma cuando sea necesario, todo de manera local.

- **Computación embebida:**

La computación embebida se refiere a sistemas de computación diseñados para realizar tareas específicas dentro de un dispositivo más grande. Estos sistemas están integrados en dispositivos que normalmente no se consideran computadoras, como electrodomésticos, vehículos, y sistemas de control industrial.

- **Relación con TinyML:**

TinyML es una forma de computación embebida donde se implementan modelos de IA en microcontroladores o sistemas de baja potencia. Estos dispositivos pueden realizar tareas como el reconocimiento de objetos, la clasificación de datos o la predicción de eventos, sin requerir un hardware avanzado.

## REFERENCIAS

- De Imagina, E. (2024, octubre 19). *Procesamiento de imágenes con OpenCV en Python*. Imaginaformacion.com; Imagina Formación. <https://imaginaformacion.com/tutoriales/opencv-en-python>
- Galicia, F. P. L. (2024, junio 13). *Guía sobre Google Colab: Aprendizaje y soluciones en programación Python*. GoDaddy Resources - LATAM; GoDaddy. <https://www.godaddy.com/resources/latam/desarrollo/google-colab-que-es-como-utilizarlo>
- Gerard, C. (2020). TensorFlow.js. *Practical Machine Learning in JavaScript*. [https://doi.org/10.1007/978-1-4842-6418-8\\_2](https://doi.org/10.1007/978-1-4842-6418-8_2)
- Guzman, H. C. (s/f). *Ejercicios «Herencia en la POO*. Hektorprofe.net. Recuperado el 19 de octubre de 2024, de <https://docs.hektorprofe.net/python/herencia-en-la-poo/ejercicios/>
- Marín, R. (2020, febrero 12). *¿Qué es OpenCV? Instalación en Python y ejemplos básicos*. Canal Informática y TICS. <https://www.inesem.es/revistadigital/informatica-y-tics/opencv/>
- Ortega, C. (2023, noviembre 7). *Modelos de machine learning: Qué son, tipos y aplicaciones*. QuestionPro. <https://www.questionpro.com/blog/es/modelos-de-machine-learning/>
- *Python desde Cero*. (s/f). Control Automático Educación; Sergio A. Castaño Giraldo. Recuperado el 19 de octubre de 2024, de <https://controlautomaticoeducacion.com/category/python-desde-cero/>
- *Python en la nube con Google Colab – Agrega*. (s/f). Agrega.com. Recuperado el 19 de octubre de 2024, de <https://www.agrega.com/blog/general/python-en-la-nube-con-google-colab/>





**RDA**  
RECURSO DIDÁCTICO PARA EL APRENDIZAJE