Introduccion a Python

Nombre: Erick Mejia

Fecha: 07/01/2026

In [2]:
```
# Comentario de python
print ("Esto es python")
```

```
Esto es python
```

1. Introduccion a Python

1.0.1 Características principales de Python

Python es un lenguaje de programación de uso general, lo que significa que puede utilizarse en muchos tipos de proyectos. No necesita ser compilado, ya que se ejecuta directamente, lo que lo hace más flexible y fácil de usar en distintos sistemas operativos.

Uno de sus mayores puntos fuertes es la facilidad de lectura del código, ya que utiliza una sintaxis clara y sencilla y es un lenguaje de alto nivel que gestiona la memoria de forma automática.

1.0.2 ¿Por qué usar Python?

Python es utilizado en muchas áreas como análisis de datos, educación, administración de sistemas, desarrollo de software y más, su popularidad se debe a su flexibilidad y su sintaxis simple. También se integra bien con otros lenguajes y herramientas, es portable entre sistemas operativos y puede optimizarse para mejorar su rendimiento cuando se trabaja con grandes volúmenes de datos.

1.0.3 ¿Qué se puede hacer con Python?

Con Python se pueden crear scripts para la administración del sistema, manipular archivos y ejecutar comandos del sistema operativo, también es posible desarrollar interfaces gráficas y aplicaciones que se comuniquen a través de internet.

Permite trabajar con bases de datos, realizar cálculos matemáticos avanzados y desarrollar aplicaciones relacionadas con inteligencia artificial y aprendizaje automático.

1.0.4 Filosofía de Python

In [1]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Python se rige por una serie de principios conocidos como El Zen de Python, los cuales promueven la simplicidad, la claridad y la legibilidad del código. Estos principios resaltan la importancia de escribir código fácil de entender, evitar complicaciones innecesarias y mantener una estructura ordenada.

1.0.5 Documentación

Python cuenta con una documentación oficial completa. Entre ella se encuentran las PEP(Python Enhancement Proposals), que definen estándares y buenas prácticas, y guías de estilo que ayudan a escribir código más ordenado.

Python Enhancement Proposals (PEP)

2. Intérprete de Python y ejecución de scripts

2.0.1 Formas de ejecutar Python

El código en Python puede ejecutarse desde la línea de comandos, usando el intérprete interactivo, mediante entornos de desarrollo (IDE) o a través de plataformas en línea como Jupyter Notebook o Google Colab.

2.0.2 ¿Qué es un intérprete?

El intérprete es un programa que se encarga de leer y ejecutar el código Python. Funciona como un intermediario entre el programa y el hardware del computador.

Existen varias versiones del intérprete, siendo la más utilizada CPython, aunque también hay otras implementaciones como Jython e IronPython.

2.0.3 Ejecución de scripts

In [3]:
```
 #Imprimir en pantalla hello"
print('Hello World')
```

```
Hello World
```

2.0.4 Cómo trabaja Python internamente

Python convierte el código escrito por el usuario en un formato intermedio llamado bytecode, el cual se ejecuta en la Máquina Virtual de Python. Este proceso permite que los programas se ejecuten de manera eficiente.

2.0.5 Uso de la línea de comandos y REPL

Permite escribir instrucciones y ver resultados inmediatamente, lo cual es útil para pruebas rápidas. Sin embargo, el código no se guarda una vez finalizada la sesión.

Este sistema interactivo sigue el modelo REPL: leer, evaluar, mostrar resultados y repetir.

2.0.6 Archivos y librerías

Python permite guardar programas en archivos y reutilizar código mediante módulos. También ofrece herramientas como pip para instalar librerías externas y ampliar sus funcionalidades.

• pip (built-in >Python3.4)

• pipenv (gestiona paquetes y entornos virtuales) o virtualenv

2.0.7 Jupyter Notebook

Jupyter permite ejecutar código de manera interactiva en celdas, facilitando la experimentación, el aprendizaje y el análisis de datos. Incluye comandos especiales y atajos que hacen el trabajo más dinámico y visual.

In [ ]:
```
# Preguntar de forma interactiva
# print?
# Usar shift + tab para hint con ayuda
# Comentar una linea
# """
# Se puede comentar un texto
# más grande para hacer descripciones detalladas con más de una línea
# """
import pandas as pd
print("Siempre podremos 'poner' los comentarios en forma de salida, para ver␣
↪resultados")
print('Siempre podremos "poner" los comentarios en forma de salida, para ver␣
↪resultados')
# universidad = 'ITQ'
# print(universidad)
```

Jupyter incluye comandos especiales que facilitan el trabajo y no forman parte del lenguaje Python como tal. Algunos de los más utilizados son:

• %run: permite ejecutar un archivo Python externo directamente desde el notebook.

• %time: mide el tiempo que tarda en ejecutarse una sola instrucción.

• %%time: calcula el tiempo de ejecución de todo el contenido de una celda.

• !: se utiliza para ejecutar comandos del sistema operativo desde el notebook.

• %%bash: permite ejecutar instrucciones del lenguaje Bash en un proceso separado.

• %%js: posibilita la ejecución de código JavaScript dentro de una celda.

In [3]:
```
ls
```

```
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: F847-3409

Directorio de C:\Users\erick\Documents\machine-2026

07/01/2026  22:20    <DIR>          .
07/01/2026  10:59    <DIR>          ..
07/01/2026  11:04    <DIR>          .ipynb_checkpoints
07/01/2026  11:37    <DIR>          img
07/01/2026  22:20             9.562 introduccion.ipynb
07/01/2026  21:17           408.131 introduccion.pdf
               2 archivos        417.693 bytes
               4 dirs  537.299.304.448 bytes libres
```

In [5]:
```
%%js
var asignatura = "Materia de Machine E-learning 1";
```

```
var codigo = 1;
alert("Esta asinatura tiene el código: " + codigo)
```

2.0.8 Rendimiento y uso de NumPy

In [4]:
```
import numpy as np
my_arr = np.arange(100) # Utilizando NumPy Arrays
my_list = list(range(100)) # Utilizando Listas
print(my_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2
3, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 4
4, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 6
5, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 8
6, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```

In [5]:
```
%time for _ in range(10): my_arr2 = my_arr * 2
```

```
CPU times: total: 15.6 ms
Wall time: 19.2 ms
```

In [6]:
```
%time for _ in range(10): my_list2 = [x * 2 for x in my_list]
```

```
CPU times: total: 250 ms
Wall time: 272 ms
```

In [ ]:
```
2.0.8 Rendimiento y uso de NumPy
```

3. Referencias

Pérez, F., & Granger, B. E. (2015). Project Jupyter: Computational narratives as the engine of collaborative data science. Retrieved from Jupyter Documentation.

Jupyter Project. (2024). Jupyter Notebook Documentation. Recuperado de https://jupyter.org/documentation

Tutorialspoint, Python Tutorial. Recuperado el 16 noviembre 2018 de https://www.tutorialspoint.com/python/

Python Software Foundation. (2023). Python Documentation. Recuperado de https://docs.python.org/3/

4. Github

https://github.com/Erick-305/machine-learning.git

In [ ]:
```

```