Guia Maestra de Python

Un Enfoque Unificado para Desarrollo de Software y Ciencia de Datos

Preparado por AcademIA
Su Learning Coach Personal

9 de noviembre de 2025

Este documento constituye una ruta de aprendizaje integral para dominar Python, fusionando las competencias esenciales del desarrollo de software y la ciencia de datos en un solo compendio teorico y practico.

Índice

1. Introduccion a Python

Python es un lenguaje de programacion de alto nivel, interpretado y de proposito general. Creado por Guido van Rossum en 1991, se ha convertido en uno de los lenguajes mas populares del mundo por su:

- **Sintaxis clara y legible**: Codigo facil de leer y escribir
- **Versatilidad**: Usado en web, ciencia de datos, IA, automatizacion, etc.
- **Gran comunidad**: Miles de librerias y recursos disponibles
- **Multiplataforma**: Funciona en Windows, Mac y Linux

Python tiene una filosofia de diseno expresada en .El Zen de Python"(PEP 20):

- Hermoso es mejor que feo
- Explicito es mejor que implicito
- Simple es mejor que complejo
- La legibilidad cuenta

Puedes verlo completo ejecutando: `import this`

2. Bloques de Construccion Esenciales

2.1 Variables y Tipos de Datos

Una **variable** es un contenedor con un nombre para almacenar datos. Cada variable tiene un **tipo de dato** que define la naturaleza de la informacion.
Tipos de datos primitivos:

- `str` (String): Texto, entre comillas. Ej: `"Hola, mundo`
- `int` (Integer): Numeros enteros. Ej: `42`
- `float` (Float): Numeros con decimales. Ej: `3.14159`
- `bool` (Boolean): Verdadero o falso. Ej: `True`, `False`
- `None`: Representa la ausencia de valor

```python
# Python es dinamicamente tipado - no necesitas declarar el tipo
nombre = "Ada Lovelace"      # str
edad = 28                    # int
altura = 1.65                # float
es_programadora = True       # bool
proyecto_actual = None       # None

# Verificar el tipo de una variable
print(type(nombre))  # <class 'str'>
print(type(edad))    # <class 'int'>
```

2.2 Operadores

Operadores en Python:
Aritmeticos:

- `+` suma, `-` resta, `*` multiplicacion, `/` division
- `//` division entera, `%` modulo (resto), `**` potencia

Comparacion:

- `==` igual, `!=` diferente
- mayor,
  menor, `>=` mayor o igual, `<=` menor o igual

Logicos:

- `and` (y logico), `or` (o logico), `not` (negacion)

2.3 Manipulacion de Strings

Los F-strings son la forma moderna y recomendada para formatear texto (Python 3.6+).

```python
nombre = "  Ada Lovelace  "
profesion = "matematica"
anio = 1843

# Metodos comunes de string
nombre_limpio = nombre.strip()      # Elimina espacios
nombre_titulo = nombre_limpio.title()  # Primera letra mayuscula
nombre_mayus = nombre_limpio.upper()   # Todo mayusculas
nombre_minus = nombre_limpio.lower()   # Todo minusculas

# F-strings (recomendado)
mensaje = f"{nombre_titulo}, la primera programadora, escribio un algoritmo en {anio}."
print(mensaje)

# Otros metodos de formato (antiguos)
mensaje2 = "{}, la primera programadora".format(nombre_titulo)
mensaje3 = "%s, la primera programadora" % nombre_titulo

# Operaciones con strings
print(len(nombre))           # Longitud
print("Ada" in nombre)       # Verificar si contiene
print(nombre.split())        # Dividir en lista
```

3. Estructuras de Datos: Organizando la Informacion

Python ofrece cuatro tipos de colecciones integradas para agrupar datos:

- **Lista**: Coleccion **ordenada** y **mutable**. Se define con corchetes `[]`. Ideal para secuencias de datos que necesitan ser modificadas.
- **Tupla**: Coleccion **ordenada** e **inmutable**. Se define con parentesis `()`. Ideal para datos que no deben cambiar.
- **Diccionario**: Coleccion de pares **clave-valor**, mutable. Se define con llaves {}. Optimizado para busquedas rapidas por clave.
- **Set**: Coleccion **no ordenada** de elementos **unicos**, mutable. Se define con llaves {}. Ideal para operaciones de conjuntos.

3.1 Listas

```python
# Crear una lista
frutas = ["manzana", "banana", "cereza"]

# Acceder a elementos (indexacion desde 0)
print(frutas[0])    # manzana
print(frutas[-1])   # cereza (ultimo elemento)

# Slicing (rebanado)
print(frutas[0:2])  # ['manzana', 'banana']

# Modificar
frutas[1] = "pera"
frutas.append("naranja")      # Agregar al final
frutas.insert(0, "uva")       # Insertar en posicion
frutas.remove("cereza")       # Eliminar por valor
frutas.pop()                  # Eliminar ultimo
frutas.pop(0)                 # Eliminar por indice

# Metodos utiles
print(len(frutas))            # Longitud
print("pera" in frutas)       # Verificar existencia
frutas.sort()                 # Ordenar
frutas.reverse()              # Invertir

# List comprehension (comprension de listas)
numeros = [1, 2, 3, 4, 5]
cuadrados = [x**2 for x in numeros]  # [1, 4, 9, 16, 25]
```

3.2 Diccionarios

```python
# Crear diccionario
persona = {
    "nombre": "Ana Garcia",
    "edad": 28,
    "ciudad": "Madrid",
    "profesion": "Data Scientist"
}

# Acceder a valores
print(persona["nombre"])        # Ana Garcia
print(persona.get("edad"))      # 28
print(persona.get("pais", "Desconocido"))  # Valor por defecto

# Modificar y agregar
persona["edad"] = 29
persona["email"] = "ana@email.com"

# Eliminar
del persona["ciudad"]
persona.pop("profesion")

# Metodos utiles
print(persona.keys())     # Todas las claves
print(persona.values())   # Todos los valores
print(persona.items())    # Pares clave-valor

# Iterar
for clave, valor in persona.items():
    print(f"{clave}: {valor}")
```

4. Control de Flujo: Tomando Decisiones y Repitiendo Tareas

4.1 Estructuras Condicionales

```python
edad = 18

if edad < 13:
    categoria = "nino"
elif edad < 18:
    categoria = "adolescente"
elif edad < 65:
    categoria = "adulto"
else:
    categoria = "adulto mayor"

print(f"Categoria: {categoria}")

# Operador ternario (condicional en una linea)
estado = "mayor" if edad >= 18 else "menor"

# Condiciones multiples
temperatura = 25
humedad = 60

if temperatura > 30 and humedad > 70:
    print("Clima caluroso y humedo")
elif temperatura > 30 or humedad > 70:
    print("Clima caluroso o humedo")
```

4.2 Bucles

```python
# Iterar sobre una lista
frutas = ["manzana", "banana", "cereza"]
for fruta in frutas:
    print(f"Me gusta la {fruta}")

# Iterar con range
for i in range(5):        # 0, 1, 2, 3, 4
    print(i)

for i in range(2, 8):     # 2, 3, 4, 5, 6, 7
    print(i)

for i in range(0, 10, 2): # 0, 2, 4, 6, 8 (paso de 2)
    print(i)

# Enumerate (obtener indice y valor)
for indice, fruta in enumerate(frutas):
    print(f"{indice}: {fruta}")

# Iterar diccionario
persona = {"nombre": "Ana", "edad": 28}
for clave, valor in persona.items():
    print(f"{clave} = {valor}")
```

```python
# Bucle while basico
contador = 0
while contador < 5:
    print(contador)
    contador += 1

# Break y continue
numero = 0
while True:
    numero += 1
    if numero == 3:
        continue  # Salta a la siguiente iteracion
    if numero > 5:
        break     # Sale del bucle
    print(numero)
```

5. Funciones: Reutilizacion y Abstraccion

Una **funcion** es un bloque de codigo reutilizable que realiza una tarea especifica. Las funciones mejoran la organizacion, legibilidad y mantenibilidad del codigo.

Componentes:

- `def`: Palabra clave para definir una funcion
- `Parametros`: Variables que recibe la funcion
- `Argumentos`: Valores pasados al llamar la funcion
- `return`: Devuelve un valor (opcional)
- `Docstring`: Documentacion de la funcion (primera linea)

```python
# Funcion simple
def saludar():
    print("Hola Mundo")

saludar()  # Llamar la funcion

# Funcion con parametros
def saludar_persona(nombre):
    """Saluda a una persona por su nombre."""
    print(f"Hola, {nombre}!")

saludar_persona("Ana")

# Funcion con return
def sumar(a, b):
    """Retorna la suma de dos numeros."""
    return a + b

resultado = sumar(5, 3)
print(resultado)  # 8

# Parametros por defecto
def potencia(base, exponente=2):
    """Calcula base elevado a exponente."""
    return base ** exponente

print(potencia(5))      # 25 (usa exponente=2)
print(potencia(5, 3))   # 125

# Argumentos por palabra clave
def crear_perfil(nombre, edad, ciudad="Desconocida"):
    return f"{nombre}, {edad} anios, de {ciudad}"

print(crear_perfil("Ana", 28))
print(crear_perfil(nombre="Luis", ciudad="Madrid", edad=35))
```

```python
# *args - numero variable de argumentos
def sumar_todos(*numeros):
    return sum(numeros)

print(sumar_todos(1, 2, 3, 4, 5))  # 15

# **kwargs - argumentos con palabra clave variable
def crear_usuario(**datos):
    for clave, valor in datos.items():
        print(f"{clave}: {valor}")

crear_usuario(nombre="Ana", edad=28, ciudad="Madrid")

# Lambda (funciones anonimas)
cuadrado = lambda x: x**2
print(cuadrado(5))  # 25

# Map, filter, reduce
numeros = [1, 2, 3, 4, 5]
cuadrados = list(map(lambda x: x**2, numeros))
pares = list(filter(lambda x: x % 2 == 0, numeros))
```

6. Gestion de Entornos con Anaconda/Conda

Un **entorno virtual** es un espacio aislado que contiene una instalacion de Python y un conjunto especifico de paquetes.
Beneficios:

- **Aislamiento**: Evita conflictos entre proyectos
- **Reproducibilidad**: Facilita compartir entornos
- **Limpieza**: Mantiene el sistema principal intacto

Comandos Esenciales de Conda:

```
# Crear entorno
conda create --name mi_proyecto python=3.10

# Activar entorno
conda activate mi_proyecto

# Instalar paquetes
conda install numpy pandas matplotlib
conda install -c conda-forge scikit-learn

# Listar entornos
conda env list

# Listar paquetes en entorno actual
conda list

# Exportar entorno
conda env export > environment.yml

# Crear entorno desde archivo
conda env create -f environment.yml

# Desactivar entorno
conda deactivate

# Eliminar entorno
conda env remove --name mi_proyecto
```

7. Desarrollo con Visual Studio Code

Visual Studio Code es un editor de codigo ligero pero potente, ideal para desarrollo en Python.

Caracteristicas clave:

- **IntelliSense**: Autocompletado inteligente
- **Linting**: Analisis de codigo (flake8, pylint)
- **Debugging**: Depurador integrado
- **Git**: Control de versiones
- **Extensions**: Extensible con miles de plugins

- **Python** (Microsoft): Extension oficial de Python
- **Pylance**: Servidor de lenguaje rapido
- **Jupyter**: Soporte para notebooks
- **GitLens**: Mejoras para Git
- **Python Docstring Generator**: Genera docstrings automaticamente

8. Manejo de Errores con Try/Except

Las excepciones son errores que ocurren durante la ejecucion. El manejo apropiado previene que el programa se detenga abruptamente.
Estructura:

- `try`: Codigo que puede generar error
- `except`: Maneja el error especifico
- `else`: Se ejecuta si no hay errores
- `finally`: Siempre se ejecuta (limpieza)

```python
# Ejemplo basico
try:
    numero = int(input("Ingresa un numero: "))
    resultado = 10 / numero
    print(f"Resultado: {resultado}")
except ValueError:
    print("Error: Debes ingresar un numero valido")
except ZeroDivisionError:
    print("Error: No puedes dividir por cero")
except Exception as e:
    print(f"Error inesperado: {e}")
else:
    print("Operacion exitosa")
finally:
    print("Fin del programa")

# Lanzar excepciones personalizadas
def dividir(a, b):
    if b == 0:
        raise ValueError("El divisor no puede ser cero")
    return a / b

try:
    resultado = dividir(10, 0)
except ValueError as e:
    print(f"Error: {e}")
```

9. Programacion Orientada a Objetos (OOP)

La POO es un paradigma que organiza el codigo en **objetos** que combinan datos (atributos) y comportamiento (metodos).
Conceptos fundamentales:

- **Clase**: Plantilla para crear objetos
- **Objeto**: Instancia de una clase
- **Atributo**: Variable que pertenece a un objeto
- **Metodo**: Funcion que pertenece a un objeto
- `__init__`: Constructor de la clase
- `self`: Referencia a la instancia actual

```python
class Persona:
    # Atributo de clase (compartido por todas las instancias)
    especie = "Homo sapiens"

    def __init__(self, nombre, edad):
        # Atributos de instancia
        self.nombre = nombre
        self.edad = edad

    # Metodo de instancia
    def saludar(self):
        return f"Hola, soy {self.nombre} y tengo {self.edad} anios"

    def cumplir_anios(self):
        self.edad += 1
        return f"Feliz cumpleanos! Ahora tengo {self.edad} anios"

    # Metodo especial (string representation)
    def __str__(self):
        return f"Persona({self.nombre}, {self.edad})"

# Crear objetos
persona1 = Persona("Ana", 28)
persona2 = Persona("Luis", 35)

# Usar metodos
print(persona1.saludar())
print(persona1.cumplir_anios())
print(persona1)  # Usa __str__

# Acceder a atributo de clase
print(Persona.especie)
```

```python
# Clase base
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hacer_sonido(self):
        pass  # Metodo abstracto

    def presentarse(self):
        return f"Soy {self.nombre}, un {self.__class__.__name__}"

# Clases derivadas (heredan de Animal)
class Perro(Animal):
    def hacer_sonido(self):
        return "Guau!"

    def traer_pelota(self):
        return f"{self.nombre} trae la pelota"

class Gato(Animal):
    def hacer_sonido(self):
        return "Miau!"

    def ronronear(self):
        return f"{self.nombre} esta ronroneando"

# Polimorfismo en accion
animales = [Perro("Rex"), Gato("Michi"), Perro("Bobby")]

for animal in animales:
    print(f"{animal.nombre}: {animal.hacer_sonido()}")
```

Objetivo: Crear un sistema de gestion de tareas usando POO.

```python
class Tarea:
    def __init__(self, descripcion, prioridad="media"):
        self.descripcion = descripcion
        self.prioridad = prioridad
        self.completada = False

    def marcar_completada(self):
        self.completada = True

    def __str__(self):
        estado = "   " if self.completada else "   "
        return f"[{estado}] {self.descripcion} (Prioridad: {self.prioridad})"

class GestorDeTareas:
    def __init__(self):
        self.tareas = []

    def agregar_tarea(self, descripcion, prioridad="media"):
        tarea = Tarea(descripcion, prioridad)
        self.tareas.append(tarea)
        print(f"Tarea agregada: {descripcion}")

    def listar_tareas(self):
        if not self.tareas:
            print("No hay tareas")
            return
        for i, tarea in enumerate(self.tareas, 1):
            print(f"{i}. {tarea}")

    def completar_tarea(self, indice):
        if 0 <= indice < len(self.tareas):
            self.tareas[indice].marcar_completada()
            print("Tarea completada")
        else:
            print("Indice invalido")

    def filtrar_por_prioridad(self, prioridad):
        filtradas = [t for t in self.tareas if t.prioridad == prioridad]
        return filtradas

# Uso
gestor = GestorDeTareas()
gestor.agregar_tarea("Estudiar Python", "alta")
gestor.agregar_tarea("Hacer ejercicio", "media")
gestor.agregar_tarea("Leer libro", "baja")
gestor.listar_tareas()
gestor.completar_tarea(0)
gestor.listar_tareas()
```

10. Trabajo con Archivos

Python permite leer y escribir archivos de manera sencilla usando la funcion `open()`.

Modos de apertura:

- `'r'`: Lectura (por defecto)
- `'w'`: Escritura (sobrescribe)
- `'a'`: Agregar (append)
- `'r+'`: Lectura y escritura
- `'b'`: Modo binario

```python
# Escribir en un archivo
with open('datos.txt', 'w') as archivo:
    archivo.write("Primera linea\n")
    archivo.write("Segunda linea\n")
    archivo.writelines(["Tercera linea\n", "Cuarta linea\n"])

# Leer archivo completo
with open('datos.txt', 'r') as archivo:
    contenido = archivo.read()
    print(contenido)

# Leer linea por linea
with open('datos.txt', 'r') as archivo:
    for linea in archivo:
        print(linea.strip())  # strip() elimina \n

# Leer todas las lineas en una lista
with open('datos.txt', 'r') as archivo:
    lineas = archivo.readlines()
    print(lineas)

# Trabajar con CSV
import csv

# Escribir CSV
datos = [
    ['Nombre', 'Edad', 'Ciudad'],
    ['Ana', 28, 'Madrid'],
    ['Luis', 35, 'Barcelona']
]

with open('personas.csv', 'w', newline='') as archivo:
    escritor = csv.writer(archivo)
    escritor.writerows(datos)

# Leer CSV
with open('personas.csv', 'r') as archivo:
    lector = csv.reader(archivo)
    for fila in lector:
        print(fila)
```

11. Analisis Numerico con NumPy

NumPy (Numerical Python) es la libreria fundamental para computacion cientifica. Su objeto principal es el `ndarray`, un arreglo multidimensional homogeneo con operaciones vectorizadas ultra rapidas.

Ventajas sobre listas:

- **Velocidad**: 10-100x mas rapido
- **Memoria**: Uso mas eficiente
- **Funcionalidad**: Operaciones matematicas avanzadas

```python
import numpy as np

# Crear arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([[1, 2, 3], [4, 5, 6]])

# Arrays especiales
zeros = np.zeros((3, 4))        # Matriz de ceros
ones = np.ones((2, 3))          # Matriz de unos
rango = np.arange(0, 10, 2)     # [0, 2, 4, 6, 8]
linspace = np.linspace(0, 1, 5) # 5 valores entre 0 y 1
aleatorio = np.random.rand(3, 3) # Matriz 3x3 aleatoria

# Operaciones vectorizadas
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

suma = a + b            # [5, 7, 9]
resta = a - b           # [-3, -3, -3]
mult = a * b            # [4, 10, 18] (elemento a elemento)
div = a / b             # [0.25, 0.4, 0.5]
potencia = a ** 2       # [1, 4, 9]

# Funciones matematicas
print(np.sin(a))        # Seno
print(np.sqrt(a))       # Raiz cuadrada
print(np.exp(a))        # Exponencial

# Estadisticas
datos = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
print(f"Media: {np.mean(datos)}")
print(f"Mediana: {np.median(datos)}")
print(f"Desviacion estandar: {np.std(datos)}")
print(f"Suma: {np.sum(datos)}")
print(f"Minimo: {np.min(datos)}")
print(f"Maximo: {np.max(datos)}")
```

```python
import numpy as np

# Array 2D
matriz = np.array([[1, 2, 3, 4],
                   [5, 6, 7, 8],
                   [9, 10, 11, 12]])

# Indexacion basica
print(matriz[0, 0])      # 1 (primera fila, primera columna)
print(matriz[1, 2])      # 7 (segunda fila, tercera columna)

# Slicing
print(matriz[0:2, 1:3])  # Filas 0-1, columnas 1-2
print(matriz[:, 2])      # Toda la tercera columna
print(matriz[1, :])      # Toda la segunda fila

# Indexacion booleana
print(matriz[matriz > 5])  # Elementos mayores que 5

# Reshape
arr = np.arange(12)
reshaped = arr.reshape(3, 4)  # Convertir a matriz 3x4

# Transpuesta
print(matriz.T)

# Concatenar arrays
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
concat_vertical = np.vstack((arr1, arr2))
concat_horizontal = np.hstack((arr1, arr2))
```

12. Manipulacion de Datos con Pandas

Pandas es la libreria esencial para analisis de datos tabulares. Sus estructuras principales son:

- **Series**: Array 1D etiquetado (una columna)
- **DataFrame**: Tabla 2D con filas y columnas etiquetadas

```python
import pandas as pd

# Crear DataFrame desde diccionario
datos = {
    'Nombre': ['Ana', 'Luis', 'Marta', 'Juan', 'Sofia'],
    'Edad': [28, 34, 29, 42, 31],
    'Ciudad': ['Madrid', 'Barcelona', 'Madrid', 'Valencia', 'Barcelona'],
    'Salario': [45000, 52000, 48000, 65000, 51000]
}
df = pd.DataFrame(datos)

# Ver informacion basica
print(df.head())        # Primeras 5 filas
print(df.tail(3))       # Ultimas 3 filas
print(df.info())        # Informacion del DataFrame
print(df.describe())    # Estadisticas descriptivas

# Seleccionar columnas
print(df['Nombre'])           # Una columna (Series)
print(df[['Nombre', 'Edad']]) # Multiples columnas (DataFrame)

# Seleccionar filas
print(df.iloc[0])       # Primera fila por indice
print(df.iloc[0:3])     # Primeras 3 filas
print(df.loc[0:2])      # Por etiqueta

# Filtrado
mayores_30 = df[df['Edad'] > 30]
madrid = df[df['Ciudad'] == 'Madrid']
salario_alto = df[df['Salario'] >= 50000]

# Filtros multiples
resultado = df[(df['Edad'] > 30) & (df['Ciudad'] == 'Barcelona')]

# Ordenar
df_ordenado = df.sort_values('Edad', ascending=False)
df_multi_orden = df.sort_values(['Ciudad', 'Edad'])

# Agregar columna
df['Antiguedad'] = [3, 5, 2, 10, 4]
df['Salario_Anual'] = df['Salario'] * 12

# Eliminar columna
df_sin_antiguedad = df.drop('Antiguedad', axis=1)

# Renombrar columnas
df_renamed = df.rename(columns={'Nombre': 'Empleado', 'Edad': 'Anos'})
```

```python
import pandas as pd

# Datos de ejemplo
ventas = pd.DataFrame({
    'Producto': ['A', 'B', 'A', 'C', 'B', 'A', 'C'],
    'Categoria': ['X', 'Y', 'X', 'Y', 'Y', 'X', 'Y'],
    'Cantidad': [10, 15, 8, 12, 20, 5, 18],
    'Precio': [100, 150, 100, 200, 150, 100, 200]
})

# GroupBy
por_producto = ventas.groupby('Producto')['Cantidad'].sum()
por_categoria = ventas.groupby('Categoria').agg({
    'Cantidad': 'sum',
    'Precio': 'mean'
})

# Pivot tables
pivot = ventas.pivot_table(
    values='Cantidad',
    index='Producto',
    columns='Categoria',
    aggfunc='sum'
)

# Valores unicos
print(ventas['Producto'].unique())
print(ventas['Producto'].value_counts())

# Manejo de valores faltantes
df_con_na = pd.DataFrame({
    'A': [1, 2, None, 4],
    'B': [5, None, None, 8],
    'C': [9, 10, 11, 12]
})

# Detectar valores nulos
print(df_con_na.isnull())
print(df_con_na.isnull().sum())

# Eliminar filas con valores nulos
df_limpio = df_con_na.dropna()

# Rellenar valores nulos
df_rellenado = df_con_na.fillna(0)
df_rellenado_media = df_con_na.fillna(df_con_na.mean())
```

```python
import pandas as pd

# Leer CSV
df = pd.read_csv('datos.csv')
df = pd.read_csv('datos.csv', sep=';')  # Otro separador
df = pd.read_csv('datos.csv', encoding='utf-8')

# Escribir CSV
df.to_csv('salida.csv', index=False)

# Leer Excel
df = pd.read_excel('datos.xlsx', sheet_name='Hoja1')

# Escribir Excel
df.to_excel('salida.xlsx', sheet_name='Resultados', index=False
    )

# Leer JSON
df = pd.read_json('datos.json')

# Escribir JSON
df.to_json('salida.json', orient='records')

# Leer SQL (requiere conexion)
import sqlite3
conn = sqlite3.connect('base_datos.db')
df = pd.read_sql_query("SELECT * FROM tabla", conn)

# Escribir SQL
df.to_sql('tabla', conn, if_exists='replace', index=False)
```

13. Visualizacion de Datos con Matplotlib

Matplotlib es la libreria base para visualizacion en Python. Proporciona control total sobre cada aspecto de una grafica.

Componentes principales:

- **Figure**: Contenedor principal
- **Axes**: Area donde se dibuja el grafico
- **Axis**: Ejes X e Y
- **Artist**: Todo lo que se ve en el grafico

```python
import matplotlib.pyplot as plt
import numpy as np

# Grafico de lineas
x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.figure(figsize=(10, 6))
plt.plot(x, y, label='sin(x)', color='blue', linewidth=2)
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('Funcion Seno')
plt.legend()
plt.grid(True)
plt.show()

# Grafico de barras
categorias = ['A', 'B', 'C', 'D']
valores = [23, 45, 56, 78]

plt.figure(figsize=(8, 6))
plt.bar(categorias, valores, color='steelblue')
plt.xlabel('Categorias')
plt.ylabel('Valores')
plt.title('Grafico de Barras')
plt.show()

# Histograma
datos = np.random.randn(1000)
plt.figure(figsize=(8, 6))
plt.hist(datos, bins=30, edgecolor='black', alpha=0.7)
plt.xlabel('Valor')
plt.ylabel('Frecuencia')
plt.title('Histograma de Distribucion Normal')
plt.show()

# Grafico de dispersion (scatter)
x = np.random.randn(100)
y = 2 * x + np.random.randn(100) * 0.5

plt.figure(figsize=(8, 6))
plt.scatter(x, y, alpha=0.5)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Grafico de Dispersion')
plt.show()
```

```python
import matplotlib.pyplot as plt
import numpy as np

# Crear figura con subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

x = np.linspace(0, 10, 100)

# Subplot 1: Seno
axes[0, 0].plot(x, np.sin(x), 'b-')
axes[0, 0].set_title('sin(x)')
axes[0, 0].grid(True)

# Subplot 2: Coseno
axes[0, 1].plot(x, np.cos(x), 'r-')
axes[0, 1].set_title('cos(x)')
axes[0, 1].grid(True)

# Subplot 3: Tangente
axes[1, 0].plot(x, np.tan(x), 'g-')
axes[1, 0].set_title('tan(x)')
axes[1, 0].set_ylim(-5, 5)
axes[1, 0].grid(True)

# Subplot 4: Exponencial
axes[1, 1].plot(x, np.exp(x), 'm-')
axes[1, 1].set_title('exp(x)')
axes[1, 1].grid(True)

plt.tight_layout()
plt.show()
```

14. Introduccion a Machine Learning con Scikit-Learn

Machine Learning es una rama de la IA que permite a las maquinas aprender patrones de datos sin ser programadas explicita mente.
Tipos principales:

- **Supervisado**: Aprendizaje con etiquetas (clasificacion, regresion)
- **No supervisado**: Sin etiquetas (clustering, reduccion dimensional)
- **Refuerzo**: Aprendizaje por recompensas

```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Generar datos de ejemplo
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X + 1 + np.random.randn(100, 1)

# Dividir en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Crear y entrenar modelo
modelo = LinearRegression()
modelo.fit(X_train, y_train)

# Predicciones
y_pred = modelo.predict(X_test)

# Evaluacion
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MSE: {mse:.2f}")
print(f"R2: {r2:.2f}")
print(f"Coeficiente: {modelo.coef_[0][0]:.2f}")
print(f"Intercepto: {modelo.intercept_[0]:.2f}")

# Visualizar
plt.scatter(X_test, y_test, color='blue', label='Datos reales')
plt.plot(X_test, y_pred, color='red', label='Prediccion')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

15. Proyectos Finales Integrados

Objetivo: Crear un sistema completo de analisis de ventas.
Requisitos:

1. Leer datos de ventas desde CSV
2. Limpiar y preparar datos (valores nulos, duplicados)
3. Calcular metricas: total ventas, promedio, top productos
4. Agrupar por categoria/mes
5. Crear visualizaciones: barras, lineas, pie charts
6. Generar reporte en formato Excel con multiples hojas

Tecnologias: Pandas, NumPy, Matplotlib, openpyxl

Objetivo: Aplicar POO para gestionar una biblioteca.
Clases necesarias:

- `Libro`: titulo, autor, ISBN, disponible
- `Usuario`: nombre, ID, libros$_p$*restados*`Biblioteca` : *catalogo*, *usuarios*, *prestamos*

Funcionalidades:

1. Agregar/eliminar libros
2. Registrar usuarios
3. Prestar/devolver libros
4. Buscar libros (por titulo, autor)
5. Generar reportes (mas prestados, usuarios activos)
6. Persistencia de datos (guardar/cargar desde archivo)

Objetivo: Crear un dashboard con analisis de datos reales.
Dataset sugerido: COVID-19, clima, finanzas, deportes
Componentes:

1. Carga y limpieza de datos
2. Analisis exploratorio (EDA)
3. Estadisticas descriptivas
4. Visualizaciones multiples
5. Analisis de tendencias temporales
6. Predicciones basicas (regresion lineal)
7. Exportar resultados

Bonus: Usar Streamlit o Dash para crear interfaz web interactiva

16. Recursos Adicionales

- **Python**: https://docs.python.org/3/
- **NumPy**: https://numpy.org/doc/
- **Pandas**: https://pandas.pydata.org/docs/
- **Matplotlib**: https://matplotlib.org/stable/contents.html
- **Scikit-Learn**: https://scikit-learn.org/stable/

- **LeetCode**: Problemas de algoritmos
- **HackerRank**: Retos de programacion
- **Kaggle**: Competencias de Data Science
- **Project Euler**: Problemas matematicos
- **CodeWars**: Katas de programacion

Fin de la Guia Maestra

Python: Desarrollo y Ciencia de Datos

Elaborado con LaTeX por AcademIA Learning Coach — 9 de noviembre de 2025