

Introducción a Python para Análisis de Datos

Bienvenidos al mundo del análisis de datos con Python

¿Qué aprenderemos hoy?

En esta primera clase exploraremos cómo Python se ha convertido en la herramienta preferida de analistas de datos a nivel mundial. Juntos daremos los primeros pasos para:

- Manipular y analizar datos de forma eficiente
- Crear reportes automáticos que ahorran horas de trabajo
- Construir pipelines ETL (Extracción, Transformación, Carga)
- Generar visualizaciones interactivas en minutos



Con **Jupyter Notebook** podrás generar visualizaciones y exportarlas como reportes HTML para compartir con tu equipo.

El panorama completo: Flujo de análisis de datos

Extracción

Lectura de datos desde múltiples fuentes:

- Archivos Excel y CSV
- Bases de datos relacionales
- APIs web y servicios en la nube
- Datos no estructurados

Transformación

Limpieza y preparación de los datos:

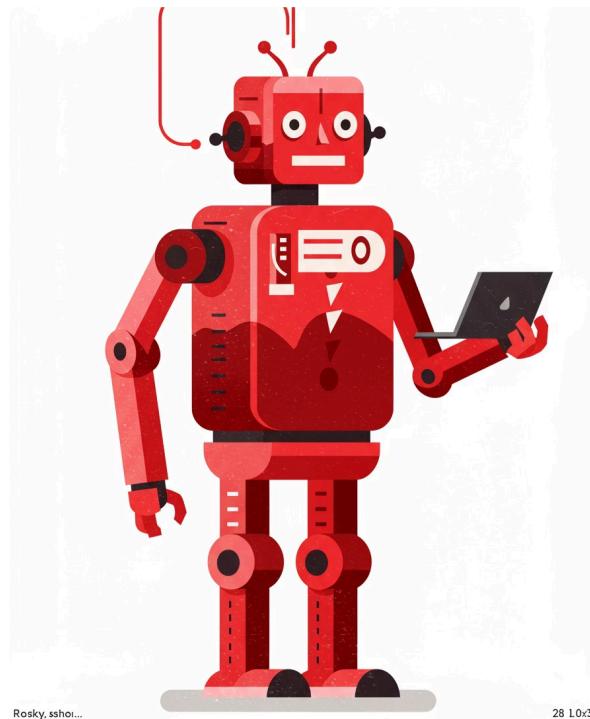
- Filtrado de registros
- Transformación de tipos
- Agregaciones y cálculos
- Combinación de fuentes

Carga

Almacenamiento de resultados:

- Exportación a archivos
- Carga en bases de datos
- Generación de reportes
- Visualizaciones interactivas

Del Notebook al script: Automatización del análisis



El camino hacia la automatización:

01

Jupyter Notebook: Entorno de desarrollo y experimentación donde probamos nuestro análisis

02

Script Python (.py): Convertimos el código validado en un programa ejecutable

03

Automatización: Se ejecuta periódicamente en un servidor o con ayuda del equipo de TI

Pregunta hipotética :

¿Es posible crear un script que cada mañana extrae datos de ventas, genera un reporte con gráficos y lo envía por correo a los directivos?

Python: Lo esencial que debes conocer

Python es un lenguaje de programación orientado a objetos donde todo es un objeto.

Esto significa que todos los elementos con los que trabajamos (números, textos, listas, incluso funciones) son objetos que tienen:

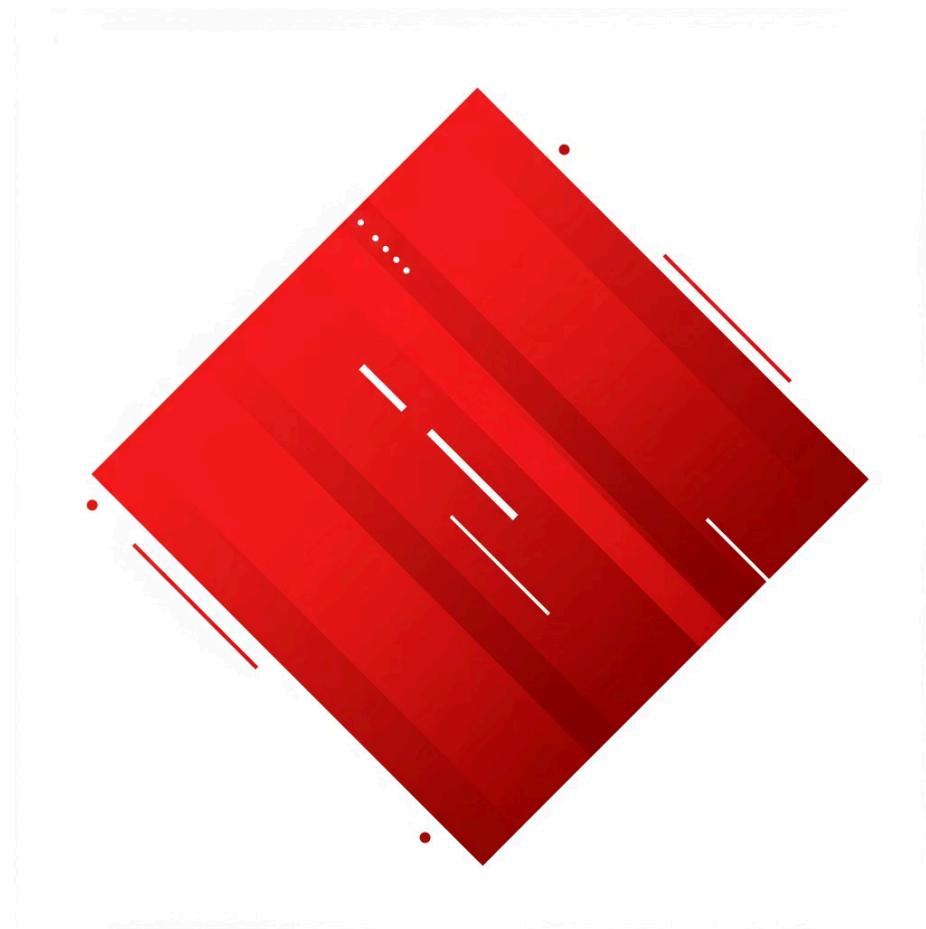
- **Propiedades:** información que contienen
- **Métodos:** acciones que pueden realizar

Ejemplo:

```
texto = "Python"
print(texto.upper()) # 'PYTHON'
print(len(texto)) # 6
print("th" in texto) # True
```

Aquí, `texto` es un objeto de tipo `string` que tiene métodos como `upper()` que podemos invocar.

La indentación: La clave del código Python



A diferencia de otros lenguajes que usan llaves {} o palabras clave como end, **Python utiliza la indentación (espacios) para definir bloques de código.**

✖ ¡Atención!

Si omites la indentación correcta, tu código generará un error.

```
if 5 > 3:  
    print("Verdadero")  
    if 10 > 8:  
        print("También verdadero")  
    print("Esto siempre se ejecuta")
```

La indentación determina qué instrucciones pertenecen a cada bloque. Este aspecto es fundamental y debes prestarle especial atención.

Tipos de datos básicos en Python

Números

Enteros (int): Números sin parte decimal

```
x = 10  
edad = -5
```

Flotantes (float): Números con parte decimal

```
y = 3.5  
precio = 99.99
```

Strings

Cadenas de texto entre comillas

```
nombre = "Ana"  
mensaje = 'Hola mundo'  
parrafo = """Texto  
en múltiples  
líneas"""
```

Son inmutables y tienen numerosos métodos útiles.

Booleanos

Representan valores de verdad

```
activo = True  
disponible = False
```

Fundamentales para control de flujo y lógica condicional en análisis de datos.

Estos tipos de datos son los bloques fundamentales con los que construiremos estructuras más complejas para nuestros análisis.

Operadores: Las herramientas para manipular datos

$\frac{f}{dx}$
Operadores Aritméticos <ul style="list-style-type: none">+ Suma: <code>5 + 3 = 8</code>- Resta: <code>5 - 3 = 2</code>* Multiplicación: <code>5 * 3 = 15</code>/ División: <code>5 / 3 = 1.667</code>% Módulo: <code>5 % 3 = 2</code>** Exponente: <code>5 ** 3 = 125</code>

>
Operadores de Comparación <ul style="list-style-type: none">== Igual a: <code>5 == 3</code> (False)!= Diferente a: <code>5 != 3</code> (True)< Menor que: <code>5 < 3</code> (False)> Mayor que: <code>5 > 3</code> (True)<= Menor o igual: <code>5 <= 5</code> (True)>= Mayor o igual: <code>5 >= 3</code> (True)


Operadores Lógicos <ul style="list-style-type: none">and Y lógico: <code>5 > 3 and 2 < 1</code> (False)or O lógico: <code>5 > 3 or 2 < 1</code> (True)not Negación: <code>not 5 > 3</code> (False) <p>Fundamentales para filtrar datos según condiciones múltiples.</p>

Estos operadores son cruciales para realizar cálculos, comparaciones y establecer condiciones en nuestros análisis de datos.

Listas: Vectores de datos flexibles

Las listas son **colecciones ordenadas y mutables** de elementos, ideales para almacenar conjuntos de datos que necesitamos manipular.

Características principales:

- Mantienen el orden de los elementos
- Permiten elementos duplicados
- Son mutables (podemos modificarlas)
- Pueden contener diferentes tipos de datos
- Se indexan desde 0

```
# Crear una lista  
valores = [10, 20, 30]
```

```
# Acceder a elementos  
print(valores[0]) # 10
```

```
# Modificar elementos  
valores[1] = 25
```

```
# Añadir elementos  
valores.append(40)
```

```
# Tamaño de la lista  
print(len(valores)) # 4
```

```
# Recorrer una lista  
for valor in valores:  
    print(valor)
```

Las listas son fundamentales en análisis de datos para almacenar colecciones de valores como series temporales, categorías o resultados de cálculos.

Tuplas y Diccionarios: Estructuras especializadas

Tuplas

Colecciones ordenadas e **inmutables**. Útiles para datos que no deben cambiar.

```
coordenadas = (10, 20)
print(coordenadas[1]) # 20

# Inmutables: no podemos modificarlas
# coordenadas[0] = 15 # Error

# Desempaquetado
x, y = coordenadas
print(x) # 10
```

Ideales para datos fijos como coordenadas, configuraciones o claves compuestas.

Diccionarios

Colecciones de pares **clave-valor**. Perfectos para datos estructurados.

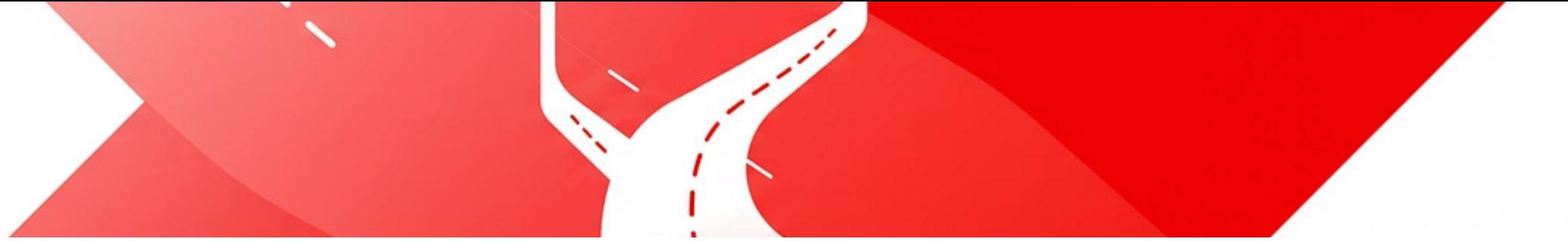
```
persona = {
    "nombre": "Ana",
    "edad": 25,
    "ciudad": "Monterrey"
}

# Acceso por clave
print(persona["edad"]) # 25

# Añadir o modificar
persona["profesión"] = "Ingeniera"

# Verificar si existe una clave
if "edad" in persona:
    print("Tiene edad")
```

Los diccionarios son especialmente importantes en análisis por su similitud con JSON y su capacidad de representar registros con múltiples atributos.



Condicionales: Toma de decisiones en Python

Las estructuras condicionales nos permiten ejecutar diferentes bloques de código según se cumplan ciertas condiciones. Son fundamentales para la lógica de análisis.

```
edad = 25
precio = 100

# Estructura if-elif-else
if edad < 18:
    descuento = 0.5 # 50% de descuento para menores
elif edad >= 65:
    descuento = 0.3 # 30% de descuento para adultos mayores
else:
    descuento = 0.1 # 10% de descuento estándar

precio_final = precio * (1 - descuento)
print(f"Precio con descuento: ${precio_final}")
```

En análisis de datos, usamos condicionales constantemente para filtrar, categorizar y transformar información según diferentes criterios.

Bucles: La repetición eficiente

Bucle for

Para **recorrer colecciones** de elementos uno por uno.

```
ciudades = ["CDMX", "Guadalajara", "Monterrey"]

for ciudad in ciudades:
    print(f"Analizando datos de {ciudad}")

# Con range() para secuencias numéricas
for i in range(1, 5): # 1, 2, 3, 4
    print(f"Iteración {i}")
```

Bucle while

Para **repetir mientras** se cumpla una condición.

```
contador = 0
total = 0

while contador < 5:
    valor = contador * 10
    total += valor
    contador += 1

print(f"Total acumulado: {total}")
```

Precaución: Asegúrate de que la condición eventualmente sea falsa para evitar bucles infinitos.

Los bucles son esenciales en análisis de datos para procesar grandes volúmenes de información, realizar cálculos iterativos y aplicar transformaciones a conjuntos de datos.

Ejemplo integrador: Análisis básico de datos



Veamos cómo integrar los conceptos aprendidos en un pequeño análisis:

```
# 1. Crear lista con datos de ventas diarias  
ventas = [1500, 2800, 1200, 3100, 2500]
```

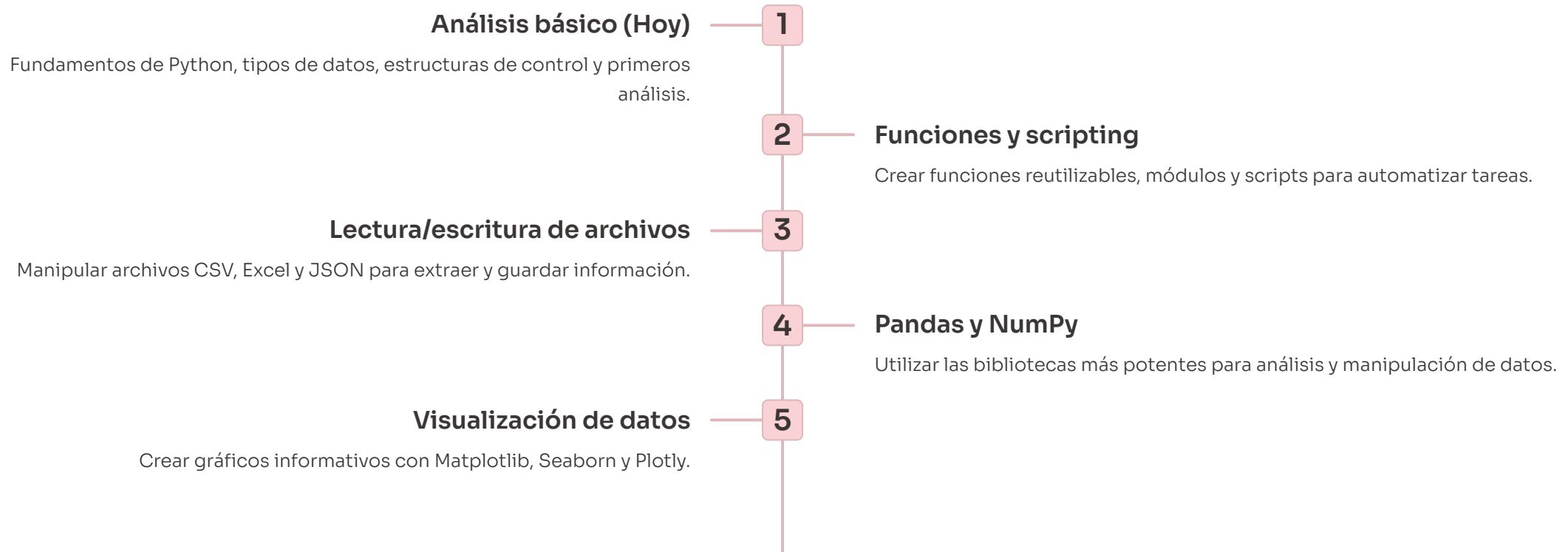
```
# 2. Calcular estadísticas básicas  
total_ventas = sum(ventas)  
promedio = total_ventas / len(ventas)  
maximo = max(ventas)  
minimo = min(ventas)
```

```
# 3. Guardar resultados en un diccionario  
resumen = {  
    "total": total_ventas,  
    "promedio": promedio,  
    "máximo": maximo,  
    "mínimo": minimo  
}
```

```
# 4. Mostrar resultados  
for clave, valor in resumen.items():  
    print(f'{clave.capitalize()}: ${valor:.2f}')
```

Este ejemplo muestra cómo podemos combinar listas, operaciones, bucles y diccionarios para realizar un análisis estadístico básico. Después podríamos exportar el Notebook a HTML para compartir el análisis.

Llevar tu análisis al siguiente nivel



ⓘ Recursos recomendados

- ✓ Documentación oficial de Python: [dhttps://www.w3schools.com/python/](https://www.w3schools.com/python/)
- ✓ Libro: "Python para análisis de datos" de Wes McKinney

¡Gracias por tu atención!

¿Preguntas?

Lo que aprendimos hoy:

- ✓ El panorama general de un flujo de análisis de datos
- ✓ Los fundamentos de Python: tipos, estructuras y control
- ✓ La importancia de la indentación y el paradigma orientado a objetos
- ✓ Cómo construir un análisis básico integrando diferentes conceptos
- ✓ El camino a seguir para convertirte en analista de datos con Python



¡Nos vemos en la próxima clase!