GUÍA MAESTRA DE

PYTHON

<De Cero a la Inteligencia Artificial/>

por

Alejandro G Vera

Guía Maestra de Python: De Cero a la Inteligencia Artificial

Alejandro G Vera

Parte I: Fundamentos Esenciales de Python

Capítulo 1: Preparando el Entorno de Desarrollo

1.1. Introducción a Python: ¿Qué es y por qué es tan popular?

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Su popularidad radica en una filosofía de diseño que prioriza la legibilidad del código y una sintaxis que permite a los desarrolladores expresar conceptos en menos líneas de código que en otros lenguajes como C++ o Java.[1] Fue creado por Guido van Rossum y lanzado por primera vez en 1991.

Las características que definen a Python y contribuyen a su adopción masiva son:

- **Sintaxis Limpia y Legible:** El código Python se asemeja al inglés, lo que facilita su aprendizaje para principiantes y aumenta la productividad de los desarrolladores experimentados.[3]
- **Interpretado:** Python ejecuta el código línea por línea, lo que simplifica la depuración. No requiere un paso de compilación explícito antes de la ejecución.
- **Versatilidad:** Es un lenguaje de propósito general, utilizado en una amplia gama de dominios, incluyendo el desarrollo web (con frameworks como Django y Flask), la ciencia de datos, el aprendizaje automático (Machine Learning), la automatización de tareas, el desarrollo de videojuegos y la computación científica.[3]

Esta combinación de simplicidad y potencia ha fomentado una comunidad global activa y solidaria, que ha desarrollado un vasto ecosistema de librerías y herramientas que extienden las capacidades del lenguaje a casi cualquier campo imaginable.

1.2. Instalación de Python: Una Decisión Fundamental

La instalación de Python no es un mero trámite, sino la primera decisión técnica que define el ecosistema de trabajo del desarrollador. La elección del método de instalación puede influir en cómo se gestionan los paquetes y los entornos virtuales en el futuro.

En Windows

Existen tres métodos principales para instalar Python en Windows, cada uno con sus propias ventajas.

- Método 1: Microsoft Store (Recomendado para principiantes)
  Este es el método más sencillo. La instalación se realiza buscando "Python" en la aplicación de la Microsoft Store y seleccionando la versión más reciente disponible (por ejemplo, Python 3.13). Este método se encarga de la configuración del PATH del sistema automáticamente y también incluye IDLE, un entorno de desarrollo integrado (IDE) simple que es útil para empezar a ejecutar comandos de Python.5
- Método 2: Instalador Oficial de Python.org (Control total)
  Para un mayor control sobre la instalación, se puede descargar el instalador directamente desde el sitio web oficial python.org.1 Durante el proceso de instalación, es crucial prestar atención a una opción específica: la casilla **"Add python.exe to PATH"**. Marcar esta opción es fundamental, ya que permite que el sistema operativo reconozca los comandos python y pip desde cualquier terminal (como el Símbolo del sistema o Windows Terminal), sin necesidad de especificar la ruta completa al ejecutable. Si no se marca esta opción, el usuario tendría que configurar manualmente las variables de entorno del sistema, un proceso que puede ser propenso a errores para los principiantes.
- Método 3: Distribución Anaconda (El camino del científico de datos)
  Anaconda es más que un simple instalador de Python; es una plataforma completa para la ciencia de datos y el aprendizaje automático. Al instalar Anaconda, no solo se obtiene Python, sino también un conjunto de librerías preinstaladas de uso común en estos campos, como NumPy, Pandas y Scikit-learn, además de herramientas como los notebooks de Jupyter y el IDE Spyder.5 Este método es ideal para quienes planean centrarse en el análisis de datos. Anaconda utiliza su propio gestor de paquetes,

conda, que está optimizado para manejar dependencias complejas y binarias comunes en las librerías científicas. La elección entre Anaconda y una instalación estándar de Python es una bifurcación temprana que define el enfoque de gestión de paquetes. Mientras que la instalación estándar utiliza pip, el gestor por defecto del ecosistema Python, Anaconda introduce conda. Un principiante que instala Anaconda y luego sigue un tutorial basado en pip puede encontrar conflictos si no comprende cómo gestionar los entornos correctamente.

En macOS

Aunque macOS suele incluir una versión preinstalada de Python, esta suele ser una versión antigua y no se recomienda su uso para el desarrollo. El método preferido es instalar una versión moderna y gestionarla de forma independiente.

- Método Recomendado: Homebrew
  Homebrew es un gestor de paquetes para macOS. Primero, se debe instalar Homebrew (si no se tiene ya) siguiendo las instrucciones de su sitio web oficial, brew.sh. Una vez instalado, instalar Python es tan simple como ejecutar un solo comando en la terminal: brew install python3. Este método no solo instala Python y pip, sino que también facilita enormemente la actualización a versiones futuras y mantiene el entorno de desarrollo limpio y organizado, separado del Python del sistema.6

En Linux (Ej. Ubuntu)

La mayoría de las distribuciones de Linux vienen con Python 3 preinstalado. Se puede verificar la versión con el comando python3 --version. Si es necesario instalarlo o actualizarlo, se utiliza el gestor de paquetes del sistema. En sistemas basados en Debian/Ubuntu, se usa apt.

Primero, se deben actualizar los repositorios de paquetes:

```
Bash
```

```
sudo apt update
```

Luego, se instala Python 3:

```
Bash
```

```
sudo apt install python3
```

Este método asegura que Python se integre correctamente con el resto del sistema operativo.[6]

1.3. Verificación de la Instalación y Primer Contacto

Independientemente del sistema operativo o del método de instalación, el paso final es verificar que todo funcione correctamente. Para ello, se debe abrir una terminal (Command Prompt o Windows Terminal en Windows, Terminal en macOS y Linux) y ejecutar uno de los siguientes comandos:

```
Bash
```

```
python --version
```

o, en algunos sistemas (especialmente macOS y Linux):

```
Bash
```

```
python3 --version
```

Si la instalación fue exitosa, la terminal mostrará la versión de Python instalada, como Python 3.13.0.[1]

Una vez verificada la versión, se puede iniciar la consola interactiva de Python (también conocida como REPL: Read-Eval-Print Loop) simplemente escribiendo python o python3 y presionando Enter. La aparición de un prompt con tres símbolos de mayor que (>>>) indica que se ha entrado en el intérprete de Python. Este es el primer entorno de experimentación directa, donde se pueden escribir líneas de código Python y ver el resultado inmediatamente.

Capítulo 2: Sintaxis y Variables: El ADN del Código

2.1. La Filosofía de Python: Legibilidad y Simplicidad

El diseño de Python está guiado por una filosofía de simplicidad y legibilidad, a menudo resumida en el "Zen de Python", un conjunto de 20 aforismos que se pueden visualizar escribiendo import this en el intérprete. Principios como "Bello es mejor que feo" y "Simple es mejor que complejo" se reflejan directamente en la sintaxis del lenguaje.[2]

- **Indentación Significativa:** La característica sintáctica más distintiva de Python es el uso de la indentación (espacios en blanco al principio de una línea) para definir bloques de código. A diferencia de muchos otros lenguajes que utilizan llaves {} o palabras clave como begin/end, en Python la estructura del programa está determinada por el nivel de sangría.[2] Esto no es una elección estilística, sino una regla sintáctica obligatoria. Esta imposición fomenta un estilo de código visualmente limpio y homogéneo en toda la comunidad, lo que facilita la lectura y el mantenimiento del código de otros, una ventaja significativa en entornos colaborativos y de código abierto.

  Python
  ```
  # La instrucción 'print' pertenece al bloque 'if' porque está indentada.
  if 5 > 2:
      print("Cinco es, en efecto, mayor que dos.") # Bloque definido por la identación [3]
  ```

- **Comentarios:** Los comentarios son texto que el intérprete de Python ignora y se

utilizan para explicar el código. Un comentario de una sola línea comienza con el símbolo de almohadilla (#). Para comentarios que abarcan múltiples líneas, se utilizan las triples comillas, ya sean dobles (""") o simples ('''). Estas últimas también se utilizan para definir "docstrings", que son cadenas de documentación para funciones y clases.[2]

2.2. Variables: Contenedores de Datos

Una variable es un nombre que se refiere a un valor. Actúan como contenedores para almacenar datos que pueden ser manipulados por el programa.

- **Creación y Asignación:** Las variables se crean en el momento en que se les asigna un valor por primera vez, utilizando el operador de asignación =.[2]
- **Tipado Dinámico:** Python es un lenguaje de tipado dinámico. Esto significa que no es necesario declarar el tipo de una variable antes de usarla; el intérprete infiere el tipo de dato automáticamente en tiempo de ejecución.[3] Además, una misma variable puede reasignarse a valores de diferentes tipos a lo largo del programa.[7]

```Python
mi_variable = 42          # Python infiere que es de tipo entero (int)
print(type(mi_variable))  # Salida: <class 'int'>
mi_variable = "Hola, mundo" # Ahora, la misma variable es de tipo cadena (str)
print(type(mi_variable))  # Salida: <class 'str'>
```

- Reglas y Convenciones de Nomenclatura:
  Los nombres de las variables deben seguir ciertas reglas 3:
  - Deben comenzar con una letra (a-z, A-Z) o un guion bajo (_).
  - El resto del nombre puede contener letras, números (0-9) y guiones bajos.
  - Son sensibles a mayúsculas y minúsculas (edad, Edad y EDAD son tres variables diferentes).
  - No pueden ser una de las palabras reservadas de Python (como if, for, class, etc.).

  Además de las reglas, la comunidad de Python sigue convenciones de estilo (conocidas como PEP 8) para mejorar la legibilidad [3]:
  - snake_case: Para nombres de variables y funciones (ej. nombre_de_usuario).
  - UPPER_CASE_SNAKE_CASE: Para constantes (ej. PI = 3.14159).
  - CamelCase: Para nombres de clases (ej. MiClaseEspecial).

2.3. Tipos de Datos Fundamentales

Python viene con varios tipos de datos incorporados que son la base para estructuras más complejas.

- **Enteros (int):** Representan números enteros, positivos o negativos, sin parte decimal. Por ejemplo: 10, -3, 0.[2]
- **Flotantes (float):** Representan números reales, es decir, números con punto decimal. Por ejemplo: 3.14, -0.001, 2.0.[2]
- **Cadenas de Texto (str):** Son secuencias de caracteres. Se definen encerrando el texto entre comillas simples (') o dobles (").[2] Python también incluye secuencias de escape, como
  \n para una nueva línea y \t para una tabulación.[7]
- **Booleanos (bool):** Representan uno de dos valores: True o False. Nótese que las primeras letras son mayúsculas. Son el resultado de operaciones de comparación y se utilizan para el control de flujo.[3]

2.4. Interacción con Tipos de Datos

- **Verificación de Tipo:** La función incorporada type() se utiliza para determinar el tipo de dato de una variable en un momento dado.[3]
- **Conversión de Tipos (Casting):** Es posible convertir explícitamente un valor de un tipo a otro utilizando funciones como int(), float() y str().[3] Esta conversión explícita es necesaria debido a que Python, a pesar de ser de tipado dinámico, es también un lenguaje de
  tipado fuerte. El tipado dinámico se refiere a *cuándo* se comprueba el tipo (en tiempo de ejecución), mientras que el tipado fuerte se refiere a *cómo* de estricto es el lenguaje con las operaciones entre tipos. Python no permite operaciones ambiguas entre tipos incompatibles. Por ejemplo, intentar sumar un número y una cadena de texto (20 + "10") resultará en un TypeError, ya que el lenguaje no sabe si la intención es una suma numérica o una concatenación de texto. Esta característica de diseño obliga al programador a ser explícito sobre sus intenciones (20 + int("10")), lo que reduce errores sutiles y mejora la claridad del código.[9]

Capítulo 3: Operadores: Las Herramientas del Programador

Los operadores son símbolos especiales en Python que realizan operaciones aritméticas o lógicas. Son los bloques de construcción fundamentales de la mayoría de las expresiones en un programa.

3.1. Operadores Aritméticos

Estos operadores se utilizan para realizar operaciones matemáticas comunes.[10]

- **Suma (+):** Suma dos operandos.
- **Resta (-):** Resta el operando derecho del izquierdo.
- **Multiplicación (*):** Multiplica dos operandos.
- **División (/):** Divide el operando izquierdo por el derecho. El resultado es siempre un float.[11]
- **División Entera (//):** Realiza una división y devuelve el resultado entero, descartando la parte decimal.[10]
- **Módulo (%):** Devuelve el resto de la división entera. Es útil para, por ejemplo, comprobar si un número es par (numero % 2 == 0).[10]
- **Exponenciación (**):** Eleva el operando izquierdo a la potencia del operando derecho.[10]

Python

```
a = 10
b = 3

print(a / b)  # Salida: 3.3333333333333335
print(a // b)  # Salida: 3
print(a % b)  # Salida: 1
print(a ** b)  # Salida: 1000
```

3.2. Operadores de Asignación

Se utilizan para asignar valores a las variables. Además del operador de asignación simple (=), Python ofrece operadores de asignación compuestos que combinan una operación aritmética con la asignación, proporcionando una sintaxis más concisa.[11]

- x += 5 es equivalente a x = x + 5
- x -= 5 es equivalente a x = x - 5
- x *= 5 es equivalente a x = x * 5

3.3. Operadores de Comparación

Estos operadores comparan dos valores y devuelven un resultado booleano: True o False. Son la base de la toma de decisiones en la programación.[10]

- **Igual a (==):** Devuelve True si los operandos son iguales.
- **Distinto de (!=):** Devuelve True si los operandos no son iguales.
- **Mayor que (>):** Devuelve True si el operando izquierdo es mayor que el derecho.
- **Menor que (<):** Devuelve True si el operando izquierdo es menor que el derecho.
- **Mayor o igual que (>=):** Devuelve True si el operando izquierdo es mayor o igual que el derecho.
- **Menor o igual que (<=):** Devuelve True si el operando izquierdo es menor o igual que el derecho.

Una característica sintáctica elegante de Python es la capacidad de encadenar operadores de comparación [11]:

Python

```
edad = 25
if 18 <= edad < 65:
    print("Es un adulto en edad laboral.")
```

3.4. Operadores Lógicos

Los operadores lógicos se utilizan para combinar expresiones condicionales.[10]

- **and:** Devuelve True si ambas expresiones son verdaderas.
- **or:** Devuelve True si al menos una de las expresiones es verdadera.
- **not:** Invierte el resultado booleano; devuelve False si el resultado es True, y viceversa.

3.5. Precedencia de Operadores

En expresiones que contienen múltiples operadores, Python los evalúa en un orden específico conocido como precedencia de operadores. Por ejemplo, la exponenciación (**) tiene una precedencia más alta que la multiplicación (*), que a su vez tiene una precedencia más alta que la suma (+).[12]

Consideremos una expresión compleja: resultado = 5 + 2 * 3 ** 2. Sin conocer la precedencia, el resultado es ambiguo. Python la evalúa de la siguiente manera:

1. Exponenciación: 3 ** 2 es 9.
2. Multiplicación: 2 * 9 es 18.
3. Suma: 5 + 18 es 23.

Si bien es útil conocer estas reglas, una práctica recomendada es utilizar paréntesis () para agrupar las operaciones y hacer la intención explícita. Esto no solo fuerza el orden de evaluación deseado, sino que también mejora drásticamente la legibilidad del código, evitando errores sutiles que pueden ser difíciles de depurar.[11] La expresión anterior podría escribirse como

resultado = 5 + (2 * (3 ** 2)) para no dejar lugar a dudas.

La siguiente tabla resume los operadores más comunes, organizados por su precedencia de mayor a menor.

Categoría	Operador	Descripción	Ejemplo	Precedencia (1=más alta)
Exponenciación	**	Eleva a la potencia	2 ** 3	1
Aritméticos (Multiplicativos)	*, /, //, %	Multiplicación, División, Div. Entera, Módulo	8 * 2, 8 / 2	2
Aritméticos (Aditivos)	+, -	Suma, Resta	5 + 3, 5 - 3	3
Comparación	==, !=, >, <, >=, <=	Comparaciones de igualdad y magnitud	x > y, x == 5	4
Lógicos	not	Negación booleana	not True	5
Lógicos	and	Y lógico	a > b and b > c	6
Lógicos	or	O lógico	a > b or b < c	7
Asignación	=, +=, -=, etc.	Asignación de valores	x = 5, x += 1	8 (más baja)

Esta tabla sirve como una referencia crucial para entender cómo Python interpreta expresiones complejas y ayuda a prevenir errores lógicos comunes.

Capítulo 4: Estructuras de Control: Dirigiendo el Flujo de tu Programa

Las estructuras de control son los mecanismos que permiten a un programa tomar decisiones y ejecutar acciones repetitivas. Dirigen el flujo de ejecución más allá de una simple secuencia de arriba hacia abajo.

4.1. Toma de Decisiones: if, elif, else

Estas sentencias permiten que el código ejecute diferentes bloques de instrucciones

basándose en si ciertas condiciones son verdaderas o falsas.

- **if simple:** Ejecuta un bloque de código únicamente si la condición especificada es True. Si es False, el bloque se omite.[13]

```python
temperatura = 30
if temperatura > 25:
    print("Hace calor. ¡No olvides hidratarte!")
```

- **if-else:** Proporciona un bloque de código alternativo que se ejecuta si la condición del if resulta ser False. Esto asegura que uno de los dos bloques siempre se ejecute.[13]

```python
numero = 15
if numero % 2 == 0:
    print(f"{numero} es un número par.")
else:
    print(f"{numero} es un número impar.")
```

- **if-elif-else:** Permite evaluar una serie de condiciones en cadena. elif es una contracción de "else if". Python evalúa las condiciones en orden: la del if primero, luego cada elif en secuencia. En cuanto encuentra una condición True, ejecuta el bloque correspondiente y omite el resto de la estructura. El bloque else final es opcional y actúa como un caso por defecto si ninguna de las condiciones anteriores fue True.[13]

```python
nota = 75
if nota >= 90:
    print("Calificación: A")
elif nota >= 80:
    print("Calificación: B")
elif nota >= 70:
    print("Calificación: C")
else:
    print("Calificación: Reprobado")
```

4.2. Bucles for: Iteración sobre Secuencias

El bucle for es la forma idiomática en Python para iterar sobre los elementos de una secuencia (como una lista, una tupla, un diccionario, un conjunto o una cadena de texto).[15] La elección entre un bucle

for y un bucle while está dictada por la naturaleza del problema: se utiliza un for cuando el número de iteraciones es **conocido** o se está recorriendo una colección finita.

- **Iterando sobre una lista:**
  Python
  ```
  frutas = ["manzana", "banana", "cereza"]
  for fruta in frutas:
      print(f"Me gusta la {fruta}.")
  ```

- **La función range():** Cuando se necesita ejecutar un bloque de código un número específico de veces, la función range() es la herramienta ideal. Genera una secuencia de números sobre la cual el bucle for puede iterar.[16]
  Python
  ```
  # Imprime los números del 0 al 4
  for i in range(5):
      print(i)
  ```

4.3. Bucles while: Repetición Basada en Condiciones

El bucle while repite un bloque de código *mientras* una condición booleana permanezca True.[16] Se utiliza cuando el número de iteraciones es

desconocido y depende de una condición que puede cambiar dinámicamente durante la ejecución del bucle, como la entrada del usuario o el resultado de un proceso.[16]

Es fundamental que la lógica dentro del bucle eventualmente haga que la condición se vuelva False. De lo contrario, se creará un **bucle infinito**, que hará que el programa se cuelgue y consuma recursos indefinidamente hasta que sea terminado manualmente (generalmente con Ctrl+C).[17]

Python

```
# Validación de entrada del usuario
contrasena_correcta = "python123"
intento = ""

while intento!= contrasena_correcta:
    intento = input("Introduce la contraseña: ")
    if intento!= contrasena_correcta:
        print("Contraseña incorrecta. Inténtalo de nuevo.")

print("Acceso concedido.")
```

4.4. Control Avanzado de Bucles

Python proporciona sentencias para un control más fino sobre la ejecución de los bucles.

- **break:** Termina la ejecución del bucle más interno de forma inmediata, y el control del programa pasa a la siguiente instrucción después del bucle.[16]
  Python
  ```
  numeros =
  for numero in numeros:
      if numero == 5:
          print("Número 5 encontrado. Saliendo del bucle.")
          break  # Sale del bucle for
      print(numero)
  ```

- **continue:** Omite el resto del código dentro del bucle para la iteración actual y pasa directamente al inicio de la siguiente iteración.[18]
  Python
  ```
  numeros =
  for numero in numeros:
      if numero % 2 == 0:
          continue  # Salta los números pares
  ```

```
    print(f"Número impar: {numero}")
```

- **La cláusula else en bucles:** Esta es una característica distintiva de Python. Un bloque else adjunto a un bucle for o while se ejecuta solo si el bucle se completa de forma natural, es decir, sin haber sido terminado por una sentencia break.[17] Esto es extremadamente útil para lógicas de búsqueda. En otros lenguajes, se necesitaría una variable de bandera ("flag") para saber si un bucle terminó porque encontró un elemento o porque agotó la colección. La sintaxis for/else de Python expresa esta intención de forma directa y legible.

```
Python
# Ejemplo de búsqueda con for/else
mi_lista =
for numero in mi_lista:
    if numero == 5:
        print("Número 5 encontrado.")
        break
else:
    # Este bloque solo se ejecuta si el 'break' nunca se activó
    print("El número 5 no está en la lista.")
```

Capítulo 5: Estructuras de Datos: Organizando la Información

5.1. Introducción a las Colecciones

Las estructuras de datos son contenedores que permiten almacenar y organizar colecciones de datos de manera eficiente.[19] Python ofrece cuatro tipos de estructuras de datos incorporadas principales, cada una con características y casos de uso distintos: Listas, Tuplas, Diccionarios y Conjuntos. La elección de la estructura de datos correcta es fundamental para escribir código eficiente y legible.

5.2. Listas: Colecciones Ordenadas y Mutables

- **Definición:** Una lista es una secuencia ordenada de elementos, encerrada entre corchetes ``. Los elementos se almacenan en un orden específico y se puede acceder a ellos mediante un índice numérico, comenzando desde 0.[21]
- **Características:**
  - **Mutables:** Se pueden modificar después de su creación. Es posible añadir, eliminar o cambiar elementos.[19]
  - **Ordenadas:** Mantienen el orden de inserción de los elementos.
  - **Permiten Duplicados:** Una lista puede contener múltiples instancias del mismo valor.
  - **Heterogéneas:** Pueden contener elementos de diferentes tipos de datos (enteros, cadenas, otras listas, etc.).[23]
- **Operaciones Comunes:**
  - Acceso: mi_lista
  - Slicing: mi_lista[1:4]
  - Añadir: mi_lista.append(elemento), mi_lista.insert(indice, elemento) [22]
  - Eliminar: del mi_lista, mi_lista.pop(), mi_lista.remove(valor) [22]
- **Casos de Uso:** Son la estructura de datos por defecto cuando se necesita una colección de elementos que puede crecer, encogerse o cambiar con el tiempo, como una lista de tareas o una serie de mediciones.[19]

Python

```
numeros_primos =
numeros_primos.append(11)  # Añade 11 al final
print(numeros_primos)      # Salida:
numeros_primos = "tres" # Modifica el segundo elemento
print(numeros_primos)      # Salida: [2, 'tres', 5, 7, 11]
```

5.3. Tuplas: Secuencias Ordenadas e Inmutables

- **Definición:** Una tupla es una secuencia ordenada de elementos, similar a una lista, pero se define con paréntesis ().[21]
- **Características:**

    - **Inmutables:** Una vez creada una tupla, no se puede modificar. No se pueden añadir, eliminar ni cambiar sus elementos.[20] Esta inmutabilidad las hace más rápidas y eficientes en memoria que las listas para almacenar datos fijos.
    - **Ordenadas:** Mantienen el orden de inserción.
    - **Permiten Duplicados:** Al igual que las listas, pueden contener elementos duplicados.
- **Creación Especial:** Para crear una tupla con un solo elemento, es necesario añadir una coma final: mi_tupla = (1,). Sin la coma, (1) se interpretaría como el número 1 entre paréntesis.[20]
- **Casos de Uso:** Se utilizan para datos que no deben cambiar, como coordenadas geográficas (latitud, longitud), constantes del programa, o como claves en un diccionario.[19]

La inmutabilidad de las tuplas es la causa directa de que puedan ser utilizadas como claves de diccionario, mientras que las listas no. Las claves de un diccionario deben ser "hashables", lo que significa que su valor no puede cambiar. Como las tuplas son inmutables, son hashables (siempre que sus elementos también lo sean), a diferencia de las listas mutables. Esta conexión entre inmutabilidad y "hashabilidad" es un concepto fundamental en la ciencia de la computación que Python implementa de forma práctica.[19]

5.4. Diccionarios: Pares Clave-Valor

- **Definición:** Un diccionario es una colección de pares clave:valor, encerrada entre llaves {}. No se basan en un orden secuencial, sino en un sistema de mapeo donde cada clave única apunta a un valor.[19]
- **Características:**
    - **Mutables:** Se pueden añadir, eliminar y modificar pares clave-valor.
    - **Ordenados (desde Python 3.7):** En versiones modernas de Python, los diccionarios mantienen el orden de inserción. Anteriormente eran no ordenados.
    - **Claves Únicas e Inmutables:** Las claves deben ser únicas dentro de un diccionario. Además, deben ser de un tipo de dato inmutable (como cadenas, números o tuplas).[20]
    - **Valores Flexibles:** Los valores pueden ser de cualquier tipo de dato, incluyendo otras estructuras de datos.
- **Operaciones Comunes:**

    - Acceso: mi_diccionario['clave']
    - Añadir/Modificar: mi_diccionario['nueva_clave'] = 'nuevo_valor'
    - Iteración: mi_diccionario.keys(), mi_diccionario.values(), mi_diccionario.items() [21]
- **Casos de Uso:** Ideales para representar objetos del mundo real (ej. un usuario con "nombre", "email", "id"), para mapear datos, o como tablas de búsqueda rápida gracias a su optimización para la recuperación de valores por clave.[19]

5.5. Conjuntos (Sets): Colecciones de Elementos Únicos

- **Definición:** Un conjunto es una colección no ordenada de elementos **únicos**, también definida con llaves {}.[19]
- **Características:**
    - **Mutables:** Se pueden añadir y eliminar elementos.
    - **No Ordenados:** No garantizan ningún orden específico de los elementos.
    - **No Permiten Duplicados:** Si se intenta añadir un elemento que ya existe, simplemente se ignora. Esta es su característica definitoria.
    - **Sin Acceso por Índice:** Debido a que no son ordenados, no se puede acceder a sus elementos mediante un índice.
- **Creación:** Para crear un conjunto vacío, se debe usar la función set(), ya que {} crea un diccionario vacío.[19]
- **Operaciones Matemáticas:** Su principal fortaleza es la capacidad de realizar operaciones de conjuntos de alta eficiencia: unión (|), intersección (&), diferencia (-) y diferencia simétrica (^).[19]
- **Casos de Uso:**
    1. **Eliminar duplicados** de una lista de forma rápida: lista_sin_duplicados = list(set(mi_lista)).
    2. **Pruebas de pertenencia:** Comprobar si un elemento existe en una colección (elemento in mi_conjunto) es extremadamente rápido.

La elección entre una lista y un conjunto para pruebas de pertenencia tiene enormes implicaciones de rendimiento. Las búsquedas en listas tienen una complejidad de tiempo O(n) (lineal), lo que significa que el tiempo de búsqueda crece con el tamaño de la lista. En cambio, las búsquedas en conjuntos son, en promedio, O(1) (constante), gracias a su implementación basada en tablas hash.[26] Para colecciones grandes, usar un conjunto puede transformar un programa inaceptablemente lento en uno

instantáneo.

Tabla Comparativa de Estructuras de Datos

La siguiente tabla resume las características clave para ayudar a elegir la estructura de datos adecuada para cada tarea.

Estructura	Sintaxis	Ordenada	Mutable	Permite Duplicados	Caso de Uso Principal
Lista	[1, 'a', 2]	Sí	Sí	Sí	Colección ordenada y modificable de elementos.
Tupla	(1, 'a', 2)	Sí	No	Sí	Almacenar datos fijos que no deben cambiar (inmutabilidad).
Diccionario	{'k1': 1, 'k2': 'a'}	Sí (desde Python 3.7)	Sí	No (claves únicas)	Mapeo de claves a valores (búsquedas rápidas por clave).
Conjunto	{1, 'a', 2}	No	Sí	No	Almacenar elementos únicos y realizar operaciones matemáticas de conjuntos.

Capítulo 6: Funciones: Creando Bloques de Código Reutilizables

Las funciones son uno de los conceptos más importantes en programación. Son bloques de código organizados y reutilizables que realizan una única acción relacionada.

6.1. La Necesidad de Funciones

El principio fundamental detrás del uso de funciones es **DRY (Don't Repeat Yourself - No te repitas)**.[27] En lugar de escribir el mismo bloque de código una y otra vez, se define una función una vez y se la llama cada vez que se necesita. Esto ofrece varias ventajas [28]:

- **Modularidad:** Divide un programa complejo en partes más pequeñas y manejables.
- **Reutilización:** El mismo código se puede usar en múltiples lugares del programa o incluso en diferentes programas.
- **Legibilidad:** Los programas son más fáciles de leer y entender cuando la lógica está organizada en funciones con nombres descriptivos.
- **Mantenimiento:** Depurar y modificar el código es más sencillo, ya que los cambios solo necesitan hacerse en un lugar: dentro de la función.

6.2. Definición y Llamada de Funciones

- **Sintaxis def:** Una función se define en Python con la palabra clave def, seguida de un nombre de función, paréntesis () que pueden contener parámetros, y dos puntos :. El cuerpo de la función debe estar indentado.[28]
  Python
  ```
  def saludar():
      print("¡Hola, bienvenido a Python!")
  ```

- **Llamada (Invocación):** Para ejecutar el código dentro de una función, se la "llama" o "invoca" escribiendo su nombre seguido de paréntesis.[28]
  Python

```
saludar()  # Esto ejecutará el print() dentro de la función.
```

6.3. Parámetros y Argumentos

Es crucial distinguir entre parámetros y argumentos [29]:

- **Parámetros:** Son las variables listadas dentro de los paréntesis en la *definición* de la función. Son marcadores de posición para los valores que la función espera recibir.
- **Argumentos:** Son los valores reales que se pasan a la función cuando esta es *llamada*.

Python

```
def saludar_a(nombre):  # 'nombre' es un parámetro
    print(f"¡Hola, {nombre}!")

saludar_a("Ana")  # "Ana" es un argumento
```

Python ofrece varias formas de pasar argumentos a las funciones:

- **Argumentos Posicionales:** Los argumentos se asignan a los parámetros según su posición. El primer argumento va al primer parámetro, y así sucesivamente.[28]
- **Argumentos de Palabra Clave (Keyword Arguments):** Se especifica explícitamente a qué parámetro se le está asignando un valor. Esto permite pasar los argumentos en cualquier orden.[29]

  Python

  ```
  def describir_mascota(tipo_animal, nombre_mascota):
      print(f"Tengo un {tipo_animal} llamado {nombre_mascota}.")

  describir_mascota(nombre_mascota="Fido", tipo_animal="perro")
  ```

- **Parámetros por Defecto:** Se puede asignar un valor por defecto a un parámetro en la definición de la función. Esto hace que el argumento sea opcional al llamar a la función. Si no se proporciona, se usará el valor por defecto.[29]

```
Python
def saludar(nombre="mundo"):
    print(f"¡Hola, {nombre}!")

saludar()        # Salida: ¡Hola, mundo!
saludar("Alice")  # Salida: ¡Hola, Alice!
```

- **Argumentos Arbitrarios (*args y **kwargs):** Para funciones que necesitan aceptar un número variable de argumentos, se utilizan *args (para argumentos posicionales) y **kwargs (para argumentos de palabra clave). *args agrupa los argumentos posicionales en una tupla, mientras que **kwargs agrupa los de palabra clave en un diccionario.[28]

6.4. La Sentencia return

La sentencia return se utiliza para que una función devuelva un valor al código que la llamó.[29]

- **Devolver un Valor:** Permite que el resultado de una función (un cálculo, un dato procesado, etc.) sea almacenado en una variable o utilizado en otras expresiones.
- **Salida Anticipada:** Cuando se ejecuta una sentencia return, la función termina inmediatamente, incluso si hay más código después de ella.[30]
- **Devolución Múltiple:** Python permite devolver múltiples valores de una manera muy limpia. Técnicamente, la función no devuelve "varios valores", sino una única tupla que contiene esos valores. Esta tupla puede ser desempaquetada en múltiples variables en el lado del que llama. Esta característica no es mágica, sino una consecuencia directa del manejo de tuplas en Python, donde return a, b es azúcar sintáctico para return (a, b).[29]

```
Python
def operaciones_basicas(a, b):
    suma = a + b
    resta = a - b
    return suma, resta  # Devuelve una tupla (suma, resta)

resultado_suma, resultado_resta = operaciones_basicas(10, 5)
print(f"Suma: {resultado_suma}, Resta: {resultado_resta}")
```

- **Retorno Implícito de None:** Si una función llega a su fin sin encontrar una

sentencia return, devuelve implícitamente el valor especial None.[29]

6.5. Ámbito de las Variables (Scope)

El ámbito de una variable determina en qué partes del programa se puede acceder a ella.

- **Variables Locales:** Se definen dentro de una función y solo existen y pueden ser accedidas dentro de esa función.[7]
- **Variables Globales:** Se definen en el cuerpo principal del programa (fuera de todas las funciones) y son accesibles desde cualquier lugar del código, tanto dentro como fuera de las funciones.[7]
- **La palabra clave global:** Para modificar una variable global desde dentro de una función, se debe declarar explícitamente con la palabra clave global. Sin embargo, se debe usar con precaución, ya que el uso excesivo de variables globales puede hacer que el código sea difícil de entender y depurar, ya que rompe la encapsulación de la función.[7]

6.6. Funciones Anónimas (Lambda)

Python permite la creación de pequeñas funciones anónimas (sin nombre) utilizando la palabra clave lambda. Una función lambda puede tomar cualquier número de argumentos, pero solo puede tener una expresión. Son sintácticamente restringidas pero muy útiles en situaciones donde se necesita una función simple por un corto período, como argumento para funciones de orden superior como map(), filter() o sorted().[29]

Python

```
# Función normal
def doble(x):
    return x * 2
```

```
# Función lambda equivalente
doble_lambda = lambda x: x * 2

print(doble(5))        # Salida: 10
print(doble_lambda(5))  # Salida: 10
```

Parte II: Python Intermedio y Programación Orientada a Objetos

Capítulo 7: Programación Orientada a Objetos (POO): Un Nuevo Paradigma

7.1. ¿Qué es la POO?

La Programación Orientada a Objetos (POO) es un paradigma de programación que organiza el software en torno a "objetos" en lugar de "acciones" y "lógica". Un objeto agrupa datos (conocidos como **atributos**) y los comportamientos o funciones que operan sobre esos datos (conocidos como **métodos**).[4] Este enfoque contrasta con la programación procedural, que se centra en una secuencia de tareas o rutinas a ejecutar.[4]

La idea central de la POO es modelar entidades del mundo real (como un coche, una persona o una cuenta bancaria) como objetos de software. En Python, esta filosofía se lleva al extremo: prácticamente todo en el lenguaje es un objeto, desde los números y las cadenas de texto hasta las propias funciones.[4] El poder de la POO radica en que permite a los programadores crear sus propios tipos de datos personalizados, extendiendo el lenguaje para que "hable" el idioma del dominio del problema que se está resolviendo.

7.2. Clases y Objetos: Planos e Instancias

Los dos conceptos fundamentales de la POO son las clases y los objetos.

- **Clases:** Una clase es un "plano", "molde" o "plantilla" para crear objetos. Define la estructura y el comportamiento que tendrán todos los objetos de ese tipo. La clase especifica qué atributos tendrá un objeto y qué métodos podrá realizar.[32] En Python, se definen con la palabra clave
  class.
- **Objetos (Instancias):** Un objeto es una instancia concreta de una clase. Si Coche es la clase, un coche rojo específico con una matrícula determinada es un objeto de esa clase. Cada objeto tiene su propio estado (los valores de sus atributos) pero comparte el mismo comportamiento (los métodos) definido por su clase.[32] Se crea un objeto "llamando" a la clase como si fuera una función: mi_coche = Coche().

7.3. El Constructor __init__ y el Parámetro self

Cuando se crea un objeto, a menudo se necesita inicializarlo con un estado inicial (es decir, darle valores iniciales a sus atributos).

- **El método __init__():** Este es un método especial en Python, conocido como "método mágico" o "dunder" (por los dobles guiones bajos). Actúa como el constructor de la clase. Se llama automáticamente cada vez que se crea una nueva instancia de la clase. Su propósito principal es inicializar los atributos del objeto.[4]
- **El parámetro self:** self es el primer parámetro obligatorio de todos los métodos de instancia en una clase, incluido __init__(). Es una referencia a la instancia del objeto actual. A través de self, los métodos pueden acceder a los atributos y otros métodos de ese objeto específico.[35] Aunque
  self no es una palabra clave de Python, es una convención extremadamente fuerte. No usar self como primer parámetro de un método de instancia se considera una violación del estilo Pythonico. Este uso explícito de self es un rasgo distintivo de Python que hace que el modelo de objetos sea más transparente en comparación con lenguajes donde la referencia a la instancia (this) es implícita.

Python

```
class Perro:
    # El método __init__ es el constructor
    def __init__(self, nombre, edad):
        # 'self' se refiere a la instancia específica que se está creando
        # Usamos self para crear y asignar atributos a la instancia
        self.nombre = nombre
        self.edad = edad
        print(f"¡Se ha creado un perro llamado {self.nombre}!")

# Creamos dos objetos (instancias) de la clase Perro
perro1 = Perro("Fido", 5)
perro2 = Perro("Laika", 3)
```

7.4. Atributos: El Estado de un Objeto

Los atributos son las variables que pertenecen a un objeto. Representan las características o el estado de ese objeto.

- **Atributos de Instancia:** Son específicos de cada objeto. Se definen dentro del método __init__ utilizando la sintaxis self.nombre_atributo = valor.[4] Cada instancia de la clase puede tener valores diferentes para sus atributos de instancia.
- **Acceso a Atributos:** Se accede a los atributos de un objeto desde fuera de la clase utilizando la notación de punto: nombre_del_objeto.nombre_del_atributo.[34]

Python

```
print(f"{perro1.nombre} tiene {perro1.edad} años.")  # Salida: Fido tiene 5 años.
print(f"{perro2.nombre} tiene {perro2.edad} años.")  # Salida: Laika tiene 3 años.
```

7.5. Métodos: El Comportamiento de un Objeto

Los métodos son funciones que están definidas dentro de una clase y describen los comportamientos que un objeto puede realizar. Operan sobre los datos (atributos) del objeto.[29]

- **Métodos de Instancia:** Al igual que __init__, los métodos de instancia siempre toman self como su primer parámetro. Esto les permite leer y modificar los atributos de la instancia.[33]

Python

```
class Estudiante:
    def __init__(self, nombre, nota):
        self.nombre = nombre
        self.nota = nota

    # Método para imprimir la información del estudiante
    def imprimir_info(self):
        print(f"Nombre: {self.nombre}, Nota: {self.nota}")

    # Método para determinar si el estudiante ha aprobado
    def resultado(self):
        if self.nota >= 5:
            print(f"{self.nombre} ha aprobado.")
        else:
            print(f"{self.nombre} ha suspendido.")

# Creamos una instancia de Estudiante
estudiante1 = Estudiante("Pedro", 7)

# Llamamos a sus métodos
estudiante1.imprimir_info()  # Salida: Nombre: Pedro, Nota: 7
estudiante1.resultado()      # Salida: Pedro ha aprobado.
```

Este ejemplo [37] muestra cómo los métodos

imprimir_info y resultado utilizan self para acceder a los atributos nombre y nota de la instancia estudiante1 y realizar acciones basadas en ellos.

Capítulo 8: Pilares de la POO en Python

La Programación Orientada a Objetos se sustenta en tres pilares conceptuales que permiten crear software robusto, flexible y reutilizable: la herencia, el polimorfismo y el encapsulamiento.

8.1. Herencia: Reutilización de Código y Jerarquías

La herencia es un mecanismo que permite a una clase, denominada **subclase** o **clase hija**, adquirir los atributos y métodos de otra clase, denominada **superclase** o **clase padre**.[27] La principal ventaja de la herencia es la reutilización de código: se puede definir un comportamiento común en una clase padre y luego crear clases hijas más especializadas que heredan ese comportamiento y añaden el suyo propio.

- **Sintaxis:** La herencia se declara en Python especificando la clase padre entre paréntesis después del nombre de la clase hija: class ClaseHija(ClasePadre):..[27]
- **La función super():** Cuando una clase hija define su propio método __init__(), a menudo necesita también ejecutar el constructor de la clase padre para inicializar los atributos heredados. La función super() proporciona una forma de llamar a los métodos de la clase padre. Usar super().__init__(...) es la práctica recomendada para asegurar que la inicialización se realice correctamente en toda la jerarquía de herencia.[33]

Python

```
# Clase Padre
class Persona:
    def __init__(self, nombre, edad):
```

```
        self.nombre = nombre
        self.edad = edad

    def presentarse(self):
        print(f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años.")

# Clase Hija que hereda de Persona
class Empleado(Persona):
    def __init__(self, nombre, edad, salario):
        # Llamamos al constructor de la clase padre (Persona) para inicializar nombre y edad
        super().__init__(nombre, edad)
        # Añadimos un nuevo atributo específico de Empleado
        self.salario = salario

    def mostrar_salario(self):
        print(f"Mi salario es de {self.salario} euros.")

# Creamos una instancia de la clase hija
empleado1 = Empleado("Carlos", 30, 45000)

empleado1.presentarse()      # Método heredado de Persona
empleado1.mostrar_salario()  # Método propio de Empleado
```

En este ejemplo [33],

Empleado hereda el comportamiento de presentarse() de Persona sin necesidad de reescribirlo.

8.2. Polimorfismo: Múltiples Formas, Una Interfaz

El polimorfismo (del griego "muchas formas") es la capacidad de objetos de diferentes clases de responder al mismo mensaje (es decir, a la misma llamada de método) de maneras específicas para cada clase.[32] En Python, el polimorfismo se logra comúnmente a través de la herencia y la

sobrescritura de métodos (method overriding), donde una clase hija proporciona una implementación específica para un método que ya está definido en su clase

padre.[38]

El polimorfismo en Python es particularmente flexible gracias a su **tipado dinámico**. El lenguaje no verifica los tipos de los objetos en tiempo de compilación. Mientras un objeto tenga el método que se está intentando llamar, el código funcionará. Este principio se conoce como "duck typing": "si camina como un pato y grazna como un pato, entonces debe ser un pato". Esto permite que una única función pueda operar sobre objetos de clases completamente diferentes, siempre que compartan una interfaz común (es decir, tengan métodos con el mismo nombre).

Python

```
class Perro(Animal): # Suponiendo que Animal tiene un método hacer_sonido
    def hacer_sonido(self):
        print("¡Guau!")

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

def escuchar_sonido_animal(animal):
    # Esta función no necesita saber si el animal es un Perro o un Gato.
    # Solo necesita que el objeto 'animal' tenga un método 'hacer_sonido()'.
    animal.hacer_sonido()

mi_perro = Perro("Scottie", 3, "Terrier")
mi_gato = Gato("Misi", 2)

escuchar_sonido_animal(mi_perro)  # Salida: ¡Guau!
escuchar_sonido_animal(mi_gato)   # Salida: ¡Miau!
```

En este ejemplo, la función escuchar_sonido_animal es polimórfica. Puede trabajar con cualquier objeto que cumpla con el "contrato" de tener un método hacer_sonido().

8.3. Encapsulamiento: Protegiendo los Datos

El encapsulamiento es el principio de ocultar los detalles de implementación internos de un objeto y exponer solo una interfaz pública y controlada. Su objetivo es proteger los datos de un objeto contra accesos y modificaciones no autorizadas, asegurando la integridad del estado del objeto.[32]

La implementación del encapsulamiento en Python refleja una filosofía de diseño fundamental del lenguaje: "somos todos adultos que consentimos". En lugar de imponer restricciones de acceso estrictas (como las palabras clave private o protected en Java o C++), Python proporciona convenciones para *indicar* la intención del programador.

- **Atributos "Protegidos" (Convención):** Un atributo cuyo nombre comienza con un solo guion bajo (ej. _saldo) se considera "protegido". Esto es una convención que le dice a otros programadores: "Este atributo es para uso interno de la clase, no deberías modificarlo directamente desde fuera". Sin embargo, no hay ninguna restricción técnica que lo impida.
- **Atributos "Privados" (Name Mangling):** Un atributo cuyo nombre comienza con dos guiones bajos (ej. __saldo) activa un mecanismo llamado *name mangling*. Python cambia internamente el nombre del atributo para incluir el nombre de la clase (ej. _CuentaBancaria__saldo), lo que hace que sea mucho más difícil acceder a él accidentalmente desde fuera de la clase o desde una subclase. No es una privacidad estricta, sino una forma de evitar colisiones de nombres en la herencia.[33]

Python

```
class CuentaBancaria:
    def __init__(self, titular, saldo_inicial):
        self.titular = titular
        self.__saldo = saldo_inicial  # Atributo "privado"

    def depositar(self, cantidad):
        if cantidad > 0:
            self.__saldo += cantidad
```

```
            print("Depósito realizado con éxito.")
        else:
            print("La cantidad a depositar debe ser positiva.")

    def retirar(self, cantidad):
        if 0 < cantidad <= self.__saldo:
            self.__saldo -= cantidad
            print("Retiro realizado con éxito.")
        else:
            print("Cantidad inválida o saldo insuficiente.")

    def obtener_saldo(self):
        # Proporcionamos un método público para acceder al saldo de forma segura
        return self.__saldo

cuenta = CuentaBancaria("Ana", 1000)

# print(cuenta.__saldo)  # Esto daría un AttributeError. El acceso directo está ofuscado.
print(f"Saldo actual: {cuenta.obtener_saldo()}") # Forma correcta de acceder

cuenta.depositar(500)
print(f"Saldo actual: {cuenta.obtener_saldo()}")
```

En este ejemplo [33], el saldo de la cuenta está encapsulado. Solo se puede modificar a través de los métodos

depositar() y retirar(), que contienen la lógica de validación, protegiendo así la integridad de los datos.

Parte III: Aplicaciones Avanzadas y del Mundo Real

Capítulo 9: Manejo de Archivos: Persistencia de Datos

Los programas necesitan interactuar con archivos para leer datos de entrada y guardar resultados de forma persistente. Python ofrece un sistema robusto y sencillo para el manejo de archivos, desde texto plano hasta formatos estructurados como CSV y JSON.

9.1. Archivos de Texto (.txt)

El manejo de archivos de texto es una de las tareas más fundamentales.

- **Abrir y Cerrar Archivos:** La función open() es el punto de entrada. Requiere al menos dos argumentos: la ruta del archivo y el modo de apertura.[39] Los modos más comunes son:
  - 'r': Lectura (read). Es el modo por defecto. El archivo debe existir.
  - 'w': Escritura (write). Crea un nuevo archivo o sobrescribe completamente uno existente.
  - 'a': Añadir (append). Abre un archivo para escribir al final del mismo sin borrar su contenido previo.
  - 'r+': Lectura y escritura.
- **El Gestor de Contexto with:** La forma idiomática y más segura de trabajar con archivos en Python es utilizando el gestor de contexto with. Esta sintaxis garantiza que el archivo se cierre automáticamente al salir del bloque with, incluso si ocurren errores durante su procesamiento. Esto previene eficazmente las fugas de recursos, que pueden ocurrir si se olvida llamar a file.close() manualmente.[39]

```
Python
# Escritura en un archivo usando 'with'
with open('saludo.txt', 'w') as f:
    f.write('Hola, mundo de los archivos!\n')
    f.write('Esta es la segunda línea.')

# Lectura del mismo archivo
with open('saludo.txt', 'r') as f:
    contenido = f.read()
    print(contenido)
```

- **Métodos de Lectura y Escritura:**
  - f.read(): Lee todo el contenido del archivo como una única cadena de texto.

- f.readline(): Lee una sola línea del archivo.
- f.readlines(): Lee todas las líneas del archivo y las devuelve como una lista de cadenas.
- f.write(cadena): Escribe la cadena proporcionada en el archivo.

9.2. Archivos CSV (Valores Separados por Comas)

El formato CSV es un estándar de facto para almacenar datos tabulares en texto plano. Es comúnmente utilizado para importar y exportar datos de hojas de cálculo y bases de datos.[42]

- **El Módulo csv:** Python incluye un módulo csv para facilitar el trabajo con estos archivos.
  - **Lectura:** La función csv.reader permite iterar sobre las filas de un archivo CSV, donde cada fila se representa como una lista de cadenas.[42] Para una lectura más semántica,
    csv.DictReader trata la primera fila como encabezados y devuelve cada fila subsiguiente como un diccionario.[43]
    Python
    ```
    import csv

    with open('datos.csv', mode='r', newline='') as archivo_csv:
        lector_csv = csv.reader(archivo_csv)
        for fila in lector_csv:
            print(fila) # Cada 'fila' es una lista de strings
    ```
    Es importante especificar newline='' al abrir el archivo para evitar que el módulo csv inserte líneas en blanco no deseadas en algunos sistemas operativos.[43]
  - **Escritura:** csv.writer proporciona el método .writerow() para escribir una lista como una fila en el archivo CSV.[42]
    Python
    ```
    import csv

    datos_a_escribir = [['Nombre', 'Ciudad'], ['Ana', 'Madrid'],]
    with open('salida.csv', mode='w', newline='') as archivo_salida:
        escritor_csv = csv.writer(archivo_salida)
    ```

```
    escritor_csv.writerows(datos_a_escribir) # Escribe todas las filas de una vez
```

- **Alternativa con Pandas:** Para análisis de datos, la librería Pandas ofrece una forma mucho más potente y conveniente de manejar archivos CSV con pd.read_csv() y df.to_csv(), que manejan automáticamente encabezados, tipos de datos y valores faltantes.[44]

9.3. Archivos JSON (JavaScript Object Notation)

JSON es el formato de intercambio de datos predominante en la web moderna, especialmente en las APIs (Interfaces de Programación de Aplicaciones). Su sintaxis de pares clave-valor se mapea de forma natural a los diccionarios de Python.[45]

El módulo json de Python distingue entre operaciones con **cadenas de texto (strings)** y operaciones con **archivos (flujos de E/S)**. Esta separación es fundamental: las funciones con una s al final (dumps, loads) operan sobre cadenas en memoria, mientras que las que no la tienen (dump, load) interactúan directamente con objetos de archivo.[47]

- **Serialización (De Python a JSON):**
  - json.dumps(objeto): ("dump string") Convierte un objeto de Python (como un diccionario o una lista) en una **cadena** de texto con formato JSON.[45]
  - json.dump(objeto, archivo): Escribe el objeto de Python directamente en un **archivo** con formato JSON.[45]
- **Deserialización (De JSON a Python):**
  - json.loads(cadena_json): ("load string") Parsea una **cadena** de texto JSON y la convierte en un objeto de Python.[48]
  - json.load(archivo): Lee de un **archivo** que contiene datos JSON y lo convierte en un objeto de Python.[45]
- **Formato y Legibilidad:** Para hacer la salida JSON legible para los humanos ("pretty-printing"), los métodos dump() y dumps() aceptan el argumento indent, que especifica el número de espacios para la sangría, y sort_keys=True para ordenar las claves del diccionario alfabéticamente.[45]

```
Python
```

```
import json

datos_persona = {
    "nombre": "Elena",
    "edad": 34,
    "es_estudiante": False,
    "cursos":
}

# Serialización a un archivo con formato legible
with open('persona.json', 'w') as f:
    json.dump(datos_persona, f, indent=4, sort_keys=True)

# Deserialización desde el archivo
with open('persona.json', 'r') as f:
    datos_leidos = json.load(f)
    print(datos_leidos['nombre']) # Salida: Elena
```

Es crucial tener en cuenta que JSON, al ser un formato de texto universal, tiene un sistema de tipos más simple que Python. Esto puede llevar a una **pérdida de fidelidad de tipos**. Por ejemplo, las tuplas de Python se serializan como "arrays" JSON, y al deserializarse de vuelta a Python, se convierten en listas, no en tuplas. Esta diferencia es importante en aplicaciones que dependen de la inmutabilidad de las tuplas.[50]

Capítulo 10: Introducción al Data Mining con NumPy y Pandas

El Data Mining o minería de datos es el proceso de descubrir patrones y conocimiento útil a partir de grandes conjuntos de datos. En Python, el ecosistema para esta tarea se cimienta sobre dos librerías fundamentales: NumPy y Pandas.

10.1. NumPy: La Base de la Computación Numérica

NumPy (Numerical Python) es la librería principal para la computación científica en Python. Su objeto central es el ndarray (array N-dimensional), una estructura de datos altamente eficiente para almacenar y operar con datos homogéneos.[51]

- **Creación de Arrays:** Se pueden crear arrays de diversas formas, como a partir de listas de Python (np.array()) o usando funciones generadoras como np.arange(), np.zeros() o np.ones().
- **El Poder de la Vectorización:** La característica más importante de NumPy es la **vectorización**. En lugar de iterar sobre los elementos de un array con un bucle de Python, NumPy permite aplicar operaciones directamente sobre arrays completos. Estas operaciones se ejecutan en código C o Fortran compilado y optimizado, lo que las hace órdenes de magnitud más rápidas que sus equivalentes en Python puro. Este aumento de rendimiento es crucial para el procesamiento de grandes volúmenes de datos.[51]

  Python
  ```
  import numpy as np
  import time

  # Con bucle de Python
  lista_py = list(range(1000000))
  start_time = time.time()
  lista_py_cuadrado = [i**2 for i in lista_py]
  print(f"Tiempo con bucle de Python: {time.time() - start_time:.6f} segundos")

  # Con NumPy (vectorización)
  array_np = np.arange(1000000)
  start_time = time.time()
  array_np_cuadrado = array_np ** 2
  print(f"Tiempo con NumPy: {time.time() - start_time:.6f} segundos")
  ```

- **Indexación Booleana:** NumPy permite filtrar arrays utilizando arrays de booleanos. Se crea un array booleano a partir de una condición y luego se usa para seleccionar solo los elementos del array original donde el valor correspondiente en el array booleano es True.[51]

10.2. Pandas: La Navaja Suiza del Análisis de Datos

Pandas es la librería de facto para la manipulación y el análisis de datos en Python. Está construida sobre NumPy, lo que significa que hereda su rendimiento para operaciones numéricas, pero añade una capa de abstracción crucial: los índices etiquetados.[52] Esto permite manipular los datos basándose en etiquetas semánticas en lugar de posiciones numéricas, lo que hace que el código sea más intuitivo y legible.

- **Estructuras de Datos Clave:**
  - **Series:** Un array unidimensional etiquetado. Esencialmente, una columna de una tabla de datos.[52]
  - **DataFrame:** Una estructura de datos bidimensional, similar a una hoja de cálculo o una tabla de SQL, donde las columnas pueden tener diferentes tipos de datos. Es la estructura de trabajo principal en Pandas.[51]
- Lectura y Exploración de Datos:
  Pandas puede leer datos de una multitud de formatos. La función más común es pd.read_csv().51 Una vez cargados los datos en un DataFrame, el primer paso es siempre la exploración inicial (Exploratory Data Analysis - EDA).
  - df.head(): Muestra las primeras 5 filas.
  - df.info(): Proporciona un resumen técnico: número de filas, columnas, tipos de datos y valores no nulos.[51]
  - df.describe(): Calcula estadísticas descriptivas (media, desviación estándar, cuartiles, etc.) para las columnas numéricas.[51]
  - df.shape: Devuelve una tupla con las dimensiones (filas, columnas).
- **Selección y Filtrado:**
  - Selección de columnas: df['nombre_columna'] o df[['col1', 'col2']].
  - Selección de filas por etiqueta: df.loc[etiqueta_fila].
  - Selección de filas por posición entera: df.iloc[posicion_fila].
  - Filtrado condicional: df[df['columna'] > 100].
- Limpieza de Datos Básica:
  Los datos del mundo real rara vez son perfectos. Pandas ofrece herramientas para limpiarlos.
  - Manejo de valores nulos: .isnull() para detectar, .dropna() para eliminar filas/columnas con nulos, y .fillna() para rellenarlos con un valor.[51]
- Agregación de Datos con groupby():
  El método .groupby() es una de las herramientas más potentes de Pandas. Permite agrupar el DataFrame por los valores de una o más columnas y luego aplicar una función de agregación (como .mean(), .sum(), .count()) a cada grupo.53

El flujo de trabajo en análisis de datos es un ciclo iterativo: los resultados de la

exploración (.info(), .describe()) informan los siguientes pasos de limpieza. Por ejemplo, si .info() revela que una columna numérica se ha cargado como tipo object (texto), esto indica un problema de limpieza que debe abordarse antes de poder realizar cálculos.[53]

Capítulo 11: Web Scraping: Extracción de Datos de la Web

El web scraping es la técnica de extraer información de sitios web de forma automatizada. En Python, la combinación de las librerías requests y BeautifulSoup es el estándar para tareas de scraping sencillas y efectivas.

11.1. Fundamentos: HTTP y HTML

Para hacer scraping, es necesario entender dos tecnologías web básicas:

- **HTTP (Protocolo de Transferencia de Hipertexto):** Es el protocolo que usan los navegadores para solicitar páginas web a los servidores. La petición más común es la GET, que se utiliza para recuperar datos de un recurso.[54]
- **HTML (Lenguaje de Marcado de Hipertexto):** Es el lenguaje estándar para crear páginas web. Estructura el contenido mediante un sistema de **etiquetas** (como <p> para un párrafo o <a> para un enlace), que pueden tener **atributos** (como class o id) que proporcionan información adicional.[55]

11.2. Paso 1: Obtener el Contenido Web con requests

La librería requests simplifica enormemente la realización de peticiones HTTP en Python.[56]

1. **Instalación:** pip install requests
2. **Realizar una petición GET:** Se utiliza requests.get(url) para solicitar la página. El objeto de respuesta contiene información valiosa, como el código de estado. Un código de 200 significa que la petición fue exitosa.[54]

3. **Acceder al contenido:** El contenido HTML de la página se encuentra en el atributo .content (en bytes) o .text (decodificado como texto) de la respuesta.[56]

Python

```
import requests

url = 'http://quotes.toscrape.com/'
response = requests.get(url)

if response.status_code == 200:
    html_content = response.text
else:
    print(f"Error al solicitar la página: Código {response.status_code}")
```

11.3. Paso 2: Analizar (Parsear) el HTML con BeautifulSoup

Una vez obtenido el HTML, se necesita una herramienta para navegar por su estructura y extraer los datos de interés. Aquí es donde entra BeautifulSoup.

1. **Instalación:** pip install beautifulsoup4
2. **Crear el objeto "Soup":** BeautifulSoup toma el contenido HTML y un parser (analizador sintáctico) para convertirlo en un árbol de objetos Python navegable.[57] El parser 'html.parser' está incluido con Python.

   Python

   ```
   from bs4 import BeautifulSoup

   soup = BeautifulSoup(html_content, 'html.parser')
   ```

3. **Encontrar Elementos:** BeautifulSoup ofrece varios métodos para localizar elementos dentro del árbol HTML:
   - **Por etiqueta:** soup.find('h1') (encuentra el primer <h1>) o soup.find_all('p') (encuentra todos los párrafos <p>).[57]

- **Por atributos:** Se pueden añadir argumentos para filtrar por class_ (nótese el guion bajo para no confundir con la palabra clave class de Python), id, etc. Ejemplo: soup.find_all('span', class_='text').[57]
- **Con Selectores CSS:** El método .select() es una forma muy potente y concisa de encontrar elementos utilizando la misma sintaxis de selectores CSS que se usa en diseño web. Por ejemplo, soup.select('div.quote span.text') encuentra todos los con clase text que están dentro de un con clase quote.[56]

4. **Extraer Datos:** Una vez que se ha localizado un elemento (un "tag object"), se puede extraer su contenido:
   - .text o .get_text(): Devuelve el contenido textual del elemento y de todos sus descendientes.
   - elemento['atributo']: Devuelve el valor de un atributo específico, como href para un enlace <a>.

11.4. Proyecto Práctico: Scraper Básico

Un proyecto de scraping típico sigue estos pasos:

1. **Inspeccionar la página de destino:** Usar las herramientas de desarrollador del navegador (clic derecho -> Inspeccionar) para identificar las etiquetas y clases HTML que contienen los datos deseados (ej. las citas y los autores en quotes.toscrape.com).[57]
2. **Escribir el script:**
   - Importar requests y BeautifulSoup.
   - Hacer la petición GET a la URL.
   - Crear el objeto soup.
   - Usar find_all o select para localizar los contenedores de cada cita.
   - Iterar sobre los contenedores y, para cada uno, extraer el texto de la cita y el nombre del autor.
   - Almacenar los datos extraídos, por ejemplo, en una lista de diccionarios.
   - Finalmente, guardar los datos en un formato estructurado como CSV.[54]

Es crucial entender que el web scraping es un proceso inherentemente **frágil**. Su éxito depende por completo de la estructura HTML del sitio web, la cual puede cambiar en cualquier momento sin previo aviso. Un cambio en el nombre de una clase o en la estructura de las etiquetas puede romper el scraper. Por esta razón, un buen script de

scraping debe incluir un manejo de errores robusto (bloques try-except) para gestionar situaciones en las que un elemento esperado no se encuentra, evitando que el programa se detenga por completo.[57] Además, el scraping a gran escala introduce desafíos técnicos y éticos, como la necesidad de rotar proxies y user-agents para evitar ser bloqueado, y la responsabilidad de no sobrecargar los servidores del sitio web.[54]

Capítulo 12: Fundamentos de Inteligencia Artificial con Scikit-Learn

La Inteligencia Artificial (IA) y, más específicamente, el Machine Learning (ML), es uno de los campos más emocionantes donde Python brilla con luz propia. Scikit-learn es la librería fundamental para iniciarse en el ML "clásico".

12.1. Introducción al Machine Learning (ML)

El Machine Learning es una rama de la IA que se enfoca en construir sistemas que pueden aprender de los datos, identificar patrones y tomar decisiones con una mínima intervención humana.[58] Se divide principalmente en dos categorías:

- **Aprendizaje Supervisado:** El algoritmo aprende de un conjunto de datos que ha sido "etiquetado" por humanos. El objetivo es aprender una función de mapeo que pueda predecir la etiqueta de salida para datos nuevos y no vistos. Se subdivide en:
  - **Clasificación:** La etiqueta de salida es una categoría discreta (ej. "spam" o "no spam", "gato" o "perro").[58]
  - **Regresión:** La etiqueta de salida es un valor continuo (ej. el precio de una casa, la temperatura de mañana).[58]
- **Aprendizaje No Supervisado:** El algoritmo trabaja con datos que no han sido etiquetados. El objetivo es descubrir la estructura o patrones ocultos en los datos.
  - **Clustering (Agrupamiento):** Agrupa los datos en clústeres basados en su similitud (ej. segmentación de clientes por comportamiento de compra).[59]

12.2. El Ecosistema de Scikit-Learn

Scikit-learn es una librería de Python de código abierto que proporciona herramientas simples y eficientes para el análisis de datos predictivo. Es la librería de referencia para tareas de ML tradicional (no deep learning) y destaca por su API consistente, su completa documentación y su integración con el ecosistema científico de Python (NumPy, SciPy, Pandas).[59]

12.3. Flujo de Trabajo Típico de un Proyecto de ML

Un proyecto de ML con Scikit-learn sigue un flujo de trabajo bien definido:

1. **Cargar los Datos:** Scikit-learn incluye varios conjuntos de datos de ejemplo (datasets "de juguete") que son perfectos para aprender, como el dataset *Iris* para clasificación y el *Boston Housing* para regresión.[58]
2. **Separar Características (X) y Objetivo (y):** Los datos se dividen en dos partes: la matriz de **características** (X), que contiene las variables predictoras (las entradas del modelo), y el vector **objetivo** (y), que contiene la variable que se quiere predecir (la salida).[61]
3. **Dividir en Conjuntos de Entrenamiento y Prueba:** Este es un paso metodológico crucial. Los datos se dividen en un conjunto de entrenamiento (típicamente 70-80% de los datos) y un conjunto de prueba (el 20-30% restante) usando la función train_test_split.[62] El modelo se *entrena* únicamente con el conjunto de entrenamiento. El conjunto de prueba se mantiene "oculto" y se usa al final para evaluar qué tan bien generaliza el modelo a datos nuevos que nunca ha visto. Este proceso es la principal defensa contra el **sobreajuste (overfitting)**, el fenómeno por el cual un modelo memoriza los datos de entrenamiento pero es incapaz de hacer predicciones precisas sobre datos nuevos.
4. **Preprocesamiento de Datos:** Los datos del mundo real a menudo requieren limpieza y transformación antes de poder ser utilizados por un modelo de ML. Esto puede incluir el escalado de características (para que todas tengan un rango similar) o la conversión de datos categóricos (texto) en representaciones numéricas.[60]

12.4. Construyendo Modelos con Scikit-Learn: La API fit/predict

La belleza de Scikit-learn reside en su API consistente y unificada. Todos los algoritmos (llamados "estimadores") siguen el mismo patrón de uso, lo que permite experimentar con diferentes modelos con cambios mínimos en el código.

1. **Importar la clase del modelo:** Por ejemplo, from sklearn.linear_model import LinearRegression.
2. **Instanciar el modelo:** Se crea un objeto del modelo: modelo = LinearRegression().
3. **Entrenar el modelo (fit):** Se entrena el modelo pasándole los datos de entrenamiento: modelo.fit(X_train, y_train).[58] Durante este paso, el algoritmo aprende los patrones en los datos.
4. **Hacer predicciones (predict):** Una vez entrenado, se utiliza el método .predict() sobre los datos de prueba (o cualquier dato nuevo) para obtener las predicciones: predicciones = modelo.predict(X_test).[58]

Python

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn import metrics

# 1. Cargar datos (dataset Iris para clasificación)
iris = datasets.load_iris()
X = iris.data  # Características
y = iris.target # Objetivo

# 2. Dividir en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 3. Instanciar y entrenar el modelo
log_reg = LogisticRegression(max_iter=200)
```

```
log_reg.fit(X_train, y_train)

# 4. Hacer predicciones
y_pred = log_reg.predict(X_test)
```

12.5. Evaluación del Modelo (Introducción)

El último paso es evaluar el rendimiento del modelo. Esto se hace comparando las predicciones (y_pred) con las etiquetas reales del conjunto de prueba (y_test).

- Para **clasificación**, métricas comunes incluyen la **precisión (accuracy)** (porcentaje de predicciones correctas), y la **matriz de confusión**, que desglosa los aciertos y errores por clase.[62]
- Para **regresión**, una métrica común es el **Error Cuadrático Medio (Mean Squared Error - MSE)**, que mide el promedio de los errores al cuadrado entre los valores predichos y los reales.[60]

```
Python

# 5. Evaluar el modelo de clasificación
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f"Precisión del modelo: {accuracy}") # Salida: Precisión del modelo: 1.0
```

La API unificada de Scikit-learn es una poderosa capa de abstracción que oculta la complejidad matemática de los algoritmos. Esto permite a los desarrolladores centrarse en el flujo de trabajo del ML y experimentar rápidamente, cambiando de un modelo a otro (ej. de Regresión Logística a un Bosque Aleatorio) con solo cambiar la línea de importación e instanciación. Esta facilidad de uso ha sido un factor clave en la democratización del machine learning.

Conclusión

Esta guía ha trazado un camino completo a través del universo de Python, desde los primeros pasos de la instalación hasta las aplicaciones avanzadas en inteligencia artificial. Se ha demostrado que Python no es solo un lenguaje, sino un ecosistema vasto y cohesivo.

Los fundamentos —sintaxis, tipos de datos, estructuras de control y funciones— establecen una base sólida sobre la cual se construyen paradigmas más complejos como la Programación Orientada a Objetos. La POO, a su vez, permite modelar problemas del mundo real de manera intuitiva y escalable.

A partir de ahí, la guía ha explorado las aplicaciones prácticas que hacen de Python el lenguaje dominante en la era de los datos. El manejo de archivos (.txt, .csv, .json) es la puerta de entrada a la persistencia de datos. El web scraping con requests y BeautifulSoup abre el acceso a la inmensa cantidad de información disponible en la web. Finalmente, el análisis de datos con NumPy y Pandas, junto con el machine learning con Scikit-learn, proporciona las herramientas para transformar datos brutos en conocimiento, predicciones y decisiones inteligentes.

El hilo conductor a lo largo de este viaje es la filosofía de Python: simplicidad, legibilidad y un enfoque pragmático para la resolución de problemas. La consistencia de las APIs en librerías clave como Scikit-learn acelera el desarrollo y la experimentación, permitiendo a los profesionales y entusiastas pasar de una idea a un prototipo funcional con una eficiencia notable. El camino desde "cero a la IA" es un testimonio de la potencia y la accesibilidad de Python, un lenguaje que ha democratizado la programación y continúa definiendo el futuro de la tecnología.

Obras citadas

1. How to install Python on Windows? - GeeksforGeeks, fecha de acceso: junio 25, 2025, https://www.geeksforgeeks.org/how-to-install-python-on-windows/
2. Fundamentos de Python: Sintaxis, variables y estructuras de control - OpenWebinars, fecha de acceso: junio 25, 2025, https://openwebinars.net/blog/fundamentos-de-python-sintaxis-variables-y-estructuras-de-control/
3. Introducción a la sintaxis básica de Python: Variables y tipos de datos, fecha de acceso: junio 25, 2025, https://www.tdscode.dev/blog/introduction-a-la-sintaxis-basica-de-python-variables-y-tipos-de-datos
4. Guía para Principiantes de la Programación Orientada a Objetos (POO) en Python - Kinsta, fecha de acceso: junio 25, 2025, https://kinsta.com/es/blog/programacion-orientada-objetos-python/
5. How to Install Python on Mac and Windows | DataCamp, fecha de acceso: junio

25, 2025, https://www.datacamp.com/blog/how-to-install-python
6. How to install Python on Windows, Linux and macOS - IONOS, fecha de acceso: junio 25, 2025, https://www.ionos.com/digitalguide/websites/web-development/install-python/
7. Sintaxis Básica y Variables en Python - Urian Viera, fecha de acceso: junio 25, 2025, https://urianviera.com/python/sintaxis-basica-en-python
8. PYTHON course from ZERO (Complete) - YouTube, fecha de acceso: junio 25, 2025, https://www.youtube.com/watch?v=nKPbfIU442g
9. Curso COMPLETO de Python DESDE CERO para Principiantes 2025 - YouTube, fecha de acceso: junio 25, 2025, https://www.youtube.com/watch?v=TkN2i-_4N4g
10. Operadores en Python: una guía práctica - Damián De Luca, fecha de acceso: junio 25, 2025, https://damiandeluca.com.ar/operadores-en-python-una-guia-practica
11. Operadores en Python: Aritméticos, comparación, lógicos, asignación, ... - J2LOGO, fecha de acceso: junio 25, 2025, https://j2logo.com/python/tutorial/operadores-en-python/
12. Operadores de Python | DataCamp, fecha de acceso: junio 25, 2025, https://www.datacamp.com/es/tutorial/python-operators-tutorial
13. if...elif...else in Python Tutorial | DataCamp, fecha de acceso: junio 25, 2025, https://www.datacamp.com/tutorial/elif-statements-python
14. Python if, if...else Statement (With Examples) - Programiz, fecha de acceso: junio 25, 2025, https://www.programiz.com/python-programming/if-elif-else
15. Conditional and control statement | PPT - SlideShare, fecha de acceso: junio 25, 2025, https://www.slideshare.net/slideshow/conditional-and-control-statement/81096218
16. Python while Loop (With Examples) - Programiz, fecha de acceso: junio 26, 2025, https://www.programiz.com/python-programming/while-loop
17. Python while Loops: Repeating Tasks Conditionally, fecha de acceso: junio 26, 2025, https://realpython.com/python-while-loop/
18. Python While Loop - GeeksforGeeks, fecha de acceso: junio 26, 2025, https://www.geeksforgeeks.org/python/python-while-loop/
19. Differences and Applications of List, Tuple, Set and Dictionary in ..., fecha de acceso: junio 26, 2025, https://www.geeksforgeeks.org/python/differences-and-applications-of-list-tuple-set-and-dictionary-in-python/
20. Lists, Tuples, Dictionaries, and Sets · HonKit - Computer Science Department, fecha de acceso: junio 26, 2025, https://cs.du.edu/~intropython/byte-of-python/data_structures.html
21. Python: Listas, Tuplas y diccionarios - Pontia tech, fecha de acceso: junio 25, 2025, https://www.pontia.tech/python-listas-tuplas-diccionario/
22. Data Structures in Python | List, Tuple, Dict, Sets, Stack, Queue - Edureka, fecha de acceso: junio 26, 2025, https://www.edureka.co/blog/data-structures-in-python/
23. Python Data Structures: Lists, Dictionaries, Sets, Tuples - Dataquest, fecha de

acceso: junio 26, 2025, https://www.dataquest.io/blog/data-structures-in-python/
24. ¿Cuándo usar conjuntos, tuplas, listas o diccionarios? : r/learnpython - Reddit, fecha de acceso: junio 25, 2025, https://www.reddit.com/r/learnpython/comments/jn4hfy/when_to_use_sets_tuples_lists_or_disctionaries/?tl=es-es
25. Python: Listas | Tuplas | Diccionarios. - Platzi, fecha de acceso: junio 25, 2025, https://platzi.com/tutoriales/4227-python-fundamentos/24974-python-listas-tuplas-diccionarios/
26. When should I use a list, dictionary, tuple, or set in Python? - Reddit, fecha de acceso: junio 26, 2025, https://www.reddit.com/r/learnprogramming/comments/1j4i759/when_should_i_use_a_list_dictionary_tuple_or_set/
27. Herencia en Python, fecha de acceso: junio 25, 2025, https://ellibrodepython.com/herencia-en-python
28. ¿Qué son las funciones de Python y cómo se utilizan? - EBAC, fecha de acceso: junio 25, 2025, https://ebac.mx/blog/funciones-de-python
29. Funciones de Python: Cómo invocar y escribir funciones | DataCamp, fecha de acceso: junio 25, 2025, https://www.datacamp.com/es/tutorial/functions-python-tutorial
30. Dominando las Funciones de Python | Tutoriales de Programación - LabEx, fecha de acceso: junio 25, 2025, https://labex.io/es/tutorials/python-python-function-fundamentals-79
31. Valores de retorno de funciones en Python - Luis Llamas, fecha de acceso: junio 25, 2025, https://www.luisllamas.es/python-retorno-funcion/
32. Dominando Python: Estructuras de datos y POO | OpenWebinars, fecha de acceso: junio 25, 2025, https://openwebinars.net/blog/python-estructuras-datos-poo/
33. POO en Python - DEV Community, fecha de acceso: junio 25, 2025, https://dev.to/maxwellnewage/poo-en-python-15ae
34. Dominando las Clases y Objetos en Python | LabEx, fecha de acceso: junio 25, 2025, https://labex.io/es/tutorials/python-classes-and-objects-71
35. Programación orientada a objetos (POO) en Python: Tutorial ..., fecha de acceso: junio 25, 2025, https://www.datacamp.com/es/tutorial/python-oop-tutorial
36. Curso de POO con PYTHON desde CERO (Completo) - YouTube, fecha de acceso: junio 25, 2025, https://www.youtube.com/watch?v=HtKqSJX7VoM
37. Ejercicios en Python - POO - Lathack, fecha de acceso: junio 25, 2025, https://lathack.com/ejercicios-en-python-poo/
38. Herencia y polimorfismo en Python: Amplía la flexibilidad de tus clases, fecha de acceso: junio 25, 2025, https://apuntes.de/python/herencia-y-polimorfismo-en-python-amplia-la-flexibilidad-de-tus-clases/
39. Trabajo con archivos en Python: lectura y escritura simplificada, fecha de acceso: junio 25, 2025, https://apuntes.de/python/trabajo-con-archivos-en-python-lectura-y-escritura-simplificada/

40. Trabajar con archivos de texto en Python - Programming Historian, fecha de acceso: junio 25, 2025, https://programminghistorian.org/es/lecciones/trabajar-con-archivos-de-texto
41. Cómo cerrar correctamente un archivo en Python - LabEx, fecha de acceso: junio 25, 2025, https://labex.io/es/tutorials/python-how-to-properly-close-a-file-in-python-398053
42. Analizando archivos CSV - Learn Python - Free Interactive Python Tutorial, fecha de acceso: junio 25, 2025, https://www.learnpython.org/es/Parsing_CSV_Files
43. Manipulación de archivos CSV en Python: Lectura y escritura de ..., fecha de acceso: junio 25, 2025, https://apuntes.de/python/manipulacion-de-archivos-csv-en-python-lectura-y-escritura-de-datos-en-formato-csvf/
44. Cómo cargar un archivo CSV en Python: Considerando sus características., fecha de acceso: junio 25, 2025, https://talkingwithdata.medium.com/c%C3%B3mo-cargar-un-archivo-csv-en-python-considerando-sus-caracter%C3%ADsticas-5cb0ca74e9d
45. Cómo Parsear JSON en Python - IPBurger.com en, fecha de acceso: junio 25, 2025, https://www.ipburger.com/es/blog/how-to-parse-json-in-python/
46. Guía para analizar datos JSON con Python - Bright Data, fecha de acceso: junio 25, 2025, https://brightdata.es/blog/procedimientos/parse-json-data-with-python
47. What is the difference between json.dumps and json.load? [closed] - Stack Overflow, fecha de acceso: junio 26, 2025, https://stackoverflow.com/questions/32911336/what-is-the-difference-between-json-dumps-and-json-load
48. Working With JSON Data in Python – Real Python, fecha de acceso: junio 26, 2025, https://realpython.com/python-json/
49. json — JSON encoder and decoder — Python 3.13.5 documentation, fecha de acceso: junio 26, 2025, https://docs.python.org/3/library/json.html
50. Tutorial de conversión de listas, tuplas y cadenas de Python a JSON - DataCamp, fecha de acceso: junio 25, 2025, https://www.datacamp.com/es/tutorial/json-data-python
51. NumPy and pandas for Data Analysis – Dataquest, fecha de acceso: junio 26, 2025, https://www.dataquest.io/tutorial/numpy-and-pandas-for-data-analysis/
52. Introduction to Pandas and NumPy - Codecademy, fecha de acceso: junio 26, 2025, https://www.codecademy.com/article/introduction-to-numpy-and-pandas
53. Python pandas Tutorial: The Ultimate Guide for Beginners - DataCamp, fecha de acceso: junio 26, 2025, https://www.datacamp.com/tutorial/pandas
54. Python Requests/BS4 Beginners Series Part 1 - First Scraper - ScrapeOps, fecha de acceso: junio 26, 2025, https://scrapeops.io/python-web-scraping-playbook/python-requests-beautifulsoup-beginners-guide/
55. Tutorial: Web Scraping with Python BeautifulSoup and Requests Libraries | by Praise James, fecha de acceso: junio 26, 2025, https://medium.com/@techwithpraisejames/web-scraping-with-beautifulsoup-an

d-requests-python-libraries-72c164b58316
56. Ultimate Guide to Web Scraping with Python Part 1: Requests and BeautifulSoup, fecha de acceso: junio 26, 2025, https://www.learndatasci.com/tutorials/ultimate-guide-web-scraping-w-python-requests-and-beautifulsoup/
57. BeautifulSoup Web Scraping: Step-By-Step Tutorial - Bright Data, fecha de acceso: junio 26, 2025, https://brightdata.com/blog/how-tos/beautiful-soup-web-scraping
58. An introduction to machine learning with scikit-learn, fecha de acceso: junio 26, 2025, https://scikit-learn.org/0.21/tutorial/basic/tutorial.html
59. Scikit-learn: A Beginner's Guide to Machine Learning in Python ..., fecha de acceso: junio 26, 2025, https://www.digitalocean.com/community/tutorials/python-scikit-learn-tutorial
60. Sklearn Linear Regression (Step-By-Step Explanation) | Sklearn Tutorial - Simplilearn.com, fecha de acceso: junio 26, 2025, https://www.simplilearn.com/tutorials/scikit-learn-tutorial/sklearn-linear-regression-with-examples
61. Learning Model Building in Scikit-learn - GeeksforGeeks, fecha de acceso: junio 26, 2025, https://www.geeksforgeeks.org/learning-model-building-scikit-learn-python-machine-learning-library/
62. Python Logistic Regression Tutorial with Sklearn & Scikit - DataCamp, fecha de acceso: junio 26, 2025, https://www.datacamp.com/tutorial/understanding-logistic-regression-python