Programación 1

# Trabajo Practico Integrador

## Gestión de datos de países en Python

Alumnos

Elias Joel Beltramino

Facundo Enrique Serrano

Fecha de entrega

24/11/2025

Indice


| | |
|---|---|
| Lista | 3 |
| Diccionarios | 4 |
| Funciones | 6 |
| Condicionales | 9 |
| Ordenamiento | 11 |
| Estadísticas Básicas | 13 |
| Archivos CSV | 14 |

# Listas

Las listas en Python son un conjunto de dato ordenados que permite almacenar datos de cualquier tipo. Son mutables y dinámicas

Los literales de lista se escriben entre corchetes [ ] y separamos sus elementos con comas. Una gran ventaja es que nos deja almacenar tipos de datos distintos

```
lista = [1, "Hola", 3.67, [1, 2, 3]]
```

**Propiedades**

- Ordenadas
- Mutable
- Indexada
- Dinámica
- Datos arbitrarios

**Acceder a lista**

Se puede acceder a un elemento de la lista usando el índice que le corresponde a ese elemento. Sabemos que los elementos inician con el índice 0 de izquierda a derecha hasta “n” por lo tanto una forma de llamar un elemento es:

Print ( lista[0]) #1

Listas anidadas

Podemos encontrar listas dentro de otras listas e incluso otra lista dentro de las dos últimas para acceder a la lista anidada debemos usar [] tantas veces como nivel de anidados tengamos

Print ( lista[3][2] ) #3

También nos permite ingresar desde la parte derecha de la lista

Print ( lista[-1][-1] ) #3

**Métodos listas**

Estos son otros métodos comunes de enumeración.

- list.append(elem): agrega un solo elemento al final de la lista.

```
l = [1, 2]
l.append(3)
print(l) #[1, 2, 3]
```

- list.insert(index, elem): inserta el elemento en el índice dado y desplaza los elementos hacia la derecha.

```
l = [1, 3]
l.insert(1, 2)
print(l) #[1, 2, 3]
```

- list.index(elem): Busca el elemento dado desde el comienzo de la lista y muestra su índice.

```
l = ["Periphery", "Intervals", "Monuments"]
print(l.index("Intervals"))
```

- list.remove(elem): busca la primera instancia del elemento determinado y la quita

```
l = [1, 2, 3]
l.remove(3)
print(l) #[1, 2]
```

- list.sort(): ordena la lista en el lugar

```
l = [3, 1, 2]
l.sort()
print(l) #[1, 2, 3]
```

- list.reverse(): Invierte la lista en el lugar

```
l = [1, 2, 3]
l.reverse()
print(l) #[3, 2, 1]
```

## Diccionarios

El diccionario de Python es una de las estructuras de datos más fundamentales y poderosas del lenguaje, diseñada para almacenar información de una manera altamente organizada y eficiente para la recuperación de datos.

**Definición General y Concepto**

Un **diccionario** en Python es una **colección de elementos** que permite almacenar datos mediante una serie de **mapeos** entre dos conjuntos de elementos interconectados: las claves (*keys*) y los valores (*values*).

- **Estructura del Dato:** Se trata de una estructura de datos que almacena su contenido en forma de pares de **llave-valor** (key: value), donde cada par constituye un **elemento** (*ítem*) del diccionario.
- **Acceso (Indexación):** A diferencia de las listas (que usan índices numéricos), los valores en un diccionario se acceden directamente a través de su **clave** concreta.

**Sintaxis y Creación**

Los diccionarios se identifican por la forma en que se encierran sus elementos:

- **Delimitadores:** Todos los elementos se encuentran encerrados en un par de corchetes curvos o llaves ({}).
- **Par Clave-Valor:** Cada elemento se escribe como clave: valor.
- **Separación:** Los diferentes pares de clave-valor se separan entre sí mediante una coma (,).

```
mi_diccionario = {"key1":<value1>,"key2":<value2>,"key3":<value3>,"key4":<value4>}
```

En el ejemplo anterior,

- El diccionario mi_diccionario contiene cuatro *pares de clave-valor,* es decir, cuatro *elementos*.
- "key1" hasta "key4" son las cuatro claves.
- Puedes usar mi_diccionario["key1"] para acceder a <value1>, y mi_diccionario["key2"] para acceder a <value2>, y así sucesivamente.

Ejemplo de sintaxis:

```
mi_diccionario = {
    "nombre": "Juan",  # "nombre" es la clave, "Juan" es el valor
    "edad": 30,        # "edad" es la clave, 30 es el valor
    "documento": "12345"
}
```

**Características Esenciales**

La potencia de los diccionarios radica en sus propiedades clave. Fundamentalmente, un diccionario se basa en el concepto de **mapeo**, permitiendo almacenar y vincular dos conjuntos de elementos: las **claves** y los **valores**.

Una característica crucial es que el acceso a los datos no se realiza por posición (índice), sino que se logra directamente a través de la **clave** asignada al valor, lo que se conoce como **indexación por clave**. Es vital que cada clave sea **única** dentro del diccionario, garantizando que no haya ambigüedad al recuperar un valor.

Además, los diccionarios son estructuras **dinámicas**, lo que significa que pueden modificarse en tiempo de ejecución: puedes añadir, eliminar o actualizar elementos fácilmente. Finalmente, son estructuras **anidables**, permitiendo que un valor a su vez sea otro diccionario, lo cual facilita la representación de datos jerárquicos y complejos.

https://ellibrodepython.com/diccionarios-en-python
https://www.datacamp.com/es/tutorial/python-dictionary-comprehension?dc_referrer=https%3A%2F%2Fwww.google.com%2F
https://www.freecodecamp.org/espanol/news/compresion-de-diccionario-en-python-explicado-con-ejemplos/. Hazme un informe con definiciones de diccionario de Python

# Funciones

Python vienen con funciones de las cuales hemos hecho uso, como por ejemplo **“int”**, que se encarga de transformar caracteres en números. Además, Python nos permite definir nuestras propias funciones utilizando la palabra reservada **“def”**

Sintaxis básica de una función

```
def nombre_funcion(parametro1, parametro2):  # 1. Definición
    """Docstring: descripción de la función"""  # 2. Documentación (opcional)
    # 3. Cuerpo de la función (indentado)
    resultado = parametro1 + parametro2
    return resultado  # 4. Retorno (opcional)
```

Estas funciones están formadas por el nombre de la función, unos argumentos de entrada, un bloque de código a ejecutar y unos parámetros de salida. Las funciones son bloques de códigos que realizan una tarea especifica y te permiten organizar tu código de manera mas clara, evitar repeticiones y facilitar su mantenimiento.

Ventajas:

Reutilización: Escribimos una función y la podemos usar muchas veces sin la necesidad de repetir el bloque

Modularidad: Nos permite organizar el programa en bloques de códigos pequeños y no tener que hacer un trozo largo de código

Mantenibilidad: Solo debemos cambiar el código de la función para que cumpla con su tarea de forma correcta y mucho más fácil de encontrar errores

Legibilidad: Código más claro

**Componentes:**

- def – Palabra clave que inicia la definición
- Nombre – Nombre de la función
- Parámetros – argumentos de entrada
- Cuerpo – código identado
- return – devuelve el valor

## Funciones con parámetros

### Parámetros posicionales

Los parámetros se leen en la posición que están definidos:

```python
def resta(a, b):
    return a - b

resta_result = resta(10, 5)
print("El resultado de la resta es:", resta_result)

resta_result = resta(5, 10)
print("El resultado de la resta es:", resta_result)

```

```
El resultado de la resta es: 5
El resultado de la resta es: -5
```

Al tratarse de parámetros posicionales, se interpretará que el primer número es la a y el segundo la b. El número de parámetros es fijo, por lo que si intentamos llamar a la función con solo uno, dará error.

**Parámetros por nombres**

Puedes especificar los parámetros por nombre, lo que hace el código más claro

```python
def crear_usuario(nombre, edad, ciudad):
    return f"{nombre}, {edad} años, de {ciudad}"

# Usando parámetros posicionales
usuario1 = crear_usuario("Ana", 25, "Madrid")

# Usando parámetros con nombre (más claro)
usuario2 = crear_usuario(nombre="Carlos", edad=30, ciudad="Barcelona")

# Puedes cambiar el orden si usas nombres
usuario3 = crear_usuario(ciudad="Valencia", nombre="Laura", edad=28)

print(usuario1)  # Ana, 25 años, de Madrid
print(usuario2)  # Carlos, 30 años, de Barcelona
print(usuario3)  # Laura, 28 años, de Valencia
```

**Parámetros con valores por defectos**

Puedes definir valores predeterminados para parámetros opcionales:

```python
def saludar(nombre, saludo="Hola"):
    """Saluda a una persona con un saludo personalizable"""
    return f"{saludo}, {nombre}!"

print(saludar("María"))  # Hola, María!
print(saludar("Pedro", "Buenos días"))  # Buenos días, Pedro!
print(saludar("Ana", saludo="Qué tal"))  # Qué tal, Ana!
```

### Sentencia return

La palabra reservada return sirve para devolver uno o varios valores desde una función, además de permitir salir de la función. Una vez que se llama a return se para la ejecución de la función y se vuelve al punto donde fue llamada. Por otro lado, puede devolver varios valores juntos separados por una coma

### Return simple

```
def sumar(a, b):
    return a + b

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

### Return múltiple

```
def calcular_estadisticas(numeros):
    """Calcula suma, promedio y máximo de una lista"""
    total = sum(numeros)
    promedio = total / len(numeros)
    maximo = max(numeros)
    return total, promedio, maximo  # Devuelve una tupla

# Desempaquetado de valores
suma, prom, max_val = calcular_estadisticas([10, 20, 30, 40, 50])
print(f"Suma: {suma}, Promedio: {prom}, Máximo: {max_val}")
# Salida: Suma: 150, Promedio: 30.0, Máximo: 50
```

### *Función sin return*

```
def imprimir_mensaje(mensaje):
    print(mensaje)
    # No hay return

resultado = imprimir_mensaje("Hola")
print(resultado)  # None
```

### *Return anticipado*

Puedes usar return para salir de una función antes de tiempo:

```
def es_par(numero):
    """Verifica si un número es par"""
    if numero % 2 == 0:
        return True  # Sale aquí si es par
    return False  # Solo se ejecuta si no es par

print(es_par(4))   # True
print(es_par(7))   # False
```

### Funciones anidadas

Puedes definir funciones dentro de otras funciones. Esto es útil para organizar código auxiliar que solo se usa dentro de una función específica.

```python
def calcular_precio_con_descuento(precio, porcentaje_descuento):
    """Calcula precio final con descuento e IVA"""

    def aplicar_descuento(p, descuento):
        """Función auxiliar interna"""
        return p * (1 - descuento / 100)

    def aplicar_iva(p):
        """Función auxiliar interna"""
        return p * 1.21

    # Usar las funciones internas
    precio_con_descuento = aplicar_descuento(precio, porcentaje_descuento)
    precio_final = aplicar_iva(precio_con_descuento)

    return precio_final

print(calcular_precio_con_descuento(100, 10))  # 108.9
```

## Condicionales

Las sentencias condicionales son estructuras de control fundamentales que permiten modificar el **flujo secuencial de ejecución** de un programa. Su función es evaluar una o varias **condiciones lógicas** (expresiones booleanas) y ejecutar bloques de código específicos solo si dichas condiciones se cumplen (es decir, se evalúan como **True**).

Principios y Sintaxis Básica

La base de las sentencias condicionales es la cláusula if.

**El Condicional Básico: if**

La sentencia if es la forma más sencilla de ejecutar código condicionalmente:

- **Propósito:** Ejecuta un bloque de instrucciones **si y solo si** la condición evaluada es verdadera (True).
- **Sintaxis:** La sentencia debe terminar con dos puntos (:) y el bloque de código a ejecutar debe ir **indentado**.

| Sintaxis | Ejemplo |
|---|---|
| if condición: | if b != 0: |
| # Bloque de código | print(a / b) |

**La Indentación:** Python utiliza la **indentación** (sangrado, usualmente de cuatro espacios) para delimitar los bloques de código que pertenecen a la sentencia condicional. A diferencia de otros lenguajes que usan llaves, en Python la indentación es obligatoria y su omisión provoca un IndentationError.

La Sentencia pass

Un bloque de código en Python **no puede estar vacío**. Si necesitas dejar un bloque if sin implementar (por ejemplo, para una tarea pendiente o para evitar un error), debes usar la

sentencia pass:

```
if a > 5:
    pass
# El programa continúa sin errores
```

**Bifurcaciones y Alternativas**

Las estructuras else y elif extienden la funcionalidad del if para manejar múltiples posibilidades y asegurar que se ejecute una acción

**Condición Doble: if ... else**

La cláusula else proporciona un camino de ejecución alternativo para cuando la condición inicial del if es falsa.

- **Propósito:** Asegura que **uno de dos bloques** se ejecute siempre. Si la condición del if es True, se ejecuta su bloque; si es False, se ejecuta el bloque del else
- **Exclusión:** El bloque if y el bloque else son **excluyentes**; solo uno se ejecutará.

| **Sintaxis** | **Ejemplo** |
|---|---|
| if condicion: | if edad < 18: |
| # Código si True | print("Menor de edad") |
| else: | else: |
| # Código si False | print("Mayor de edad") |

**Múltiples Alternativas: if ... elif ... else**

Cuando se tienen varias condiciones distintas que comprobar, se utiliza la cláusula elif (contracción de *else if*).

- **Propósito:** Permite **encadenar** múltiples condiciones. El programa comprueba las condiciones en orden, de arriba abajo.
- **Flujo:** Tan pronto como se encuentra la **primera condición** que es True (ya sea en el if o en un elif), se ejecuta su bloque de código asociado y **el resto de la estructura es ignorada** (incluyendo los elif y el else posteriores).
- **Estructura:** Puede haber un solo if al inicio, un número indefinido de elif, y un solo else opcional al final para capturar todos los casos no cubiertos.

```
x = 7
if x > 10:
    print('x es mayor que 10')
elif x < 10:  # Si el if anterior fue False, comprueba esta.
    print('x es menor que 10')
else:         # Si ninguna fue True, ejecuta este bloque.
    print('x es 10')
```

**El Operador Ternario (Condicional en una Línea)**

Python ofrece una forma concisa de escribir una sentencia simple if-else en una sola línea, conocida como **operador ternario**.

- **Propósito:** Asignar un valor a una variable o ejecutar una expresión basándose en una condición, todo en una sola línea de código.
- **Sintaxis:**

  [expresión si True] if [condición] else [expresión si False]

```
# Asigna el valor 2.0 a 'c' si b es distinto de cero, sino asigna -1.
a = 10
b = 5
c = a / b if b != 0 else -1
print(c)  # Resultado: 2.0
```

## Ordenamiento

El ordenamiento es una operación fundamental en la programación, utilizada para organizar datos de manera eficiente. Python proporciona métodos integrados sencillos para ordenar listas, así como la flexibilidad para implementar algoritmos de ordenamiento personalizados.

**Métodos Integrados de Ordenamiento en Python**

Python ofrece dos formas principales y eficientes para ordenar datos, que internamente utilizan el algoritmo Timsort (un híbrido de *merge sort* e *insertion sort*, conocido por su eficiencia):

- **list.sort()**: Este es un **método** que solo funciona con listas. Ordena la lista *en su lugar*, lo que significa que modifica la lista original y no devuelve ningún valor (devuelve None).
- **sorted()**: Esta es una **función** incorporada que funciona con cualquier tipo de iterable (listas, tuplas, cadenas, etc.) y devuelve una *nueva* lista ordenada, dejando intacto el iterable original

Ambos métodos aceptan argumentos opcionales para personalizar el orden:

- **reverse**: Un argumento booleano (por defecto es False). Si se establece en True, la lista se ordena en orden descendente.

- **key**: Un argumento que acepta una función. Esta función se aplica a cada elemento antes de hacer las comparaciones, permitiendo ordenar según un criterio específico. Por ejemplo, sorted(lista, key=len) ordenaría los elementos por su longitud

### Algoritmos de Ordenamiento Comunes

Más allá de las funciones integradas, existen diversos algoritmos de ordenamiento clásicos que se pueden implementar en Python para comprender cómo funcionan internamente.

#### Ordenamiento de Burbuja (*Bubble Sort*)

El *Bubble Sort* es un algoritmo simple que funciona revisando repetidamente la lista y comparando pares de elementos adyacentes, intercambiándolos si están en el orden incorrecto. Este proceso se repite hasta que la lista entera está ordenada. Su implementación implica dos bucles anidados:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
        if not swapped: break

# Dada una lista desordenada
x = [9, 8, 7, 5, 4, 3, 0]

# La ordenamos utilizando bubble sort
bubble_sort(x)
print(x)
# [0, 3, 4, 5, 7, 8, 9]
```

Aunque es fácil de entender, el *Bubble Sort* es ineficiente para listas grandes debido a su complejidad temporal.

#### Otros Algoritmos

- **Ordenamiento por Selección (*Selection Sort*)**: Funciona seleccionando repetidamente el elemento más pequeño (o más grande) de la parte no ordenada de la lista y moviéndolo a su posición correcta en la parte ordenada.
- **Ordenamiento por Inserción (*Insertion Sort*)**, **Ordenamiento por Mezcla (*Merge Sort*)**, **Ordenamiento Rápido (*Quick Sort*)**, entre otros, son algoritmos con diferentes eficiencias y casos de uso que también se pueden implementar en Python.

En resumen, Python facilita el ordenamiento eficiente mediante sort() y sorted(), mientras que la implementación de algoritmos personalizados es útil para fines educativos y para situaciones específicas donde se requiere un control manual del proceso

## Estadísticas básicas

Para calcular estadísticas básicas en Python, puedes utilizar bibliotecas potentes como Pandas, NumPy y el módulo estándar “statistics”. Estas herramientas facilitan el cálculo de medidas de tendencia central y dispersión.

En nuestro proyecto hemos implementado el modulo estándar de Python para estadísticas “statistics”. Adecuada para estadísticas matemáticas básicas sobre datos numéricos. Es ideal para conjuntos de datos pequeños o situaciones en las que no se pueden utilizar librerías externas.

Las funciones clave incluyen:

- **Media, Mediana, Moda**: statistics.mean(), statistics.median(), statistics.mode()
- **Desviación Estándar y Varianza**: statistics.stdev() (muestra) y statistics.pstdev() (población), junto con statistics.variance() y statistics.pvariance()
- **Otras medidas**: harmonic_mean(), median_grouped(), quantiles()

Para poder hacer uso de esta librería de estadísticas estándar de Python, debemos importarla al programa donde queramos utilizarla con el comando “import statistics”.

```python
import statistics

# Lista de datos de ejemplo (por ejemplo, puntuaciones de un examen)
datos = [85, 90, 78, 92, 88, 76, 95, 89, 81]

# Calcular la media (promedio)
media = statistics.mean(datos)

# Calcular la mediana (el valor central cuando los datos están ordenados)
mediana = statistics.median(datos)

# Calcular la desviación estándar (medida de dispersión)
desviacion_estandar = statistics.stdev(datos)

# Imprimir los resultados
print(f"Datos: {datos}")
print(f"Media (Promedio): {media:.2f}")
print(f"Mediana: {mediana}")
print(f"Desviación Estándar: {desviacion_estandar:.2f}")
```

```
Datos: [85, 90, 78, 92, 88, 76, 95, 89, 81]
Media (Promedio): 86.00
Mediana: 88
Desviación Estándar: 6.00
```

# Archivos csv

El formato **CSV** (*Comma Separated Values* o **Valores Separados por Comas**) es un estándar de facto para el intercambio de datos tabulares. Su principal característica es la **simplicidad**, ya que se trata de un **archivo de texto plano** que permite almacenar datos estructurados de manera universal y eficiente.

### Estructura de Datos Tabular

Un archivo CSV sigue una estructura bidimensional clara, equiparable a una tabla de base de datos o una hoja de cálculo:

- **Registro (Fila):** Cada línea del archivo representa un registro completo de datos.
- **Campo (Columna):** Los valores dentro de cada línea están separados por un carácter específico.

| Componente | Descripción |
|---|---|
| Delimitador (delimiter) | El carácter que separa los campos de datos. Aunque la coma (,) es el predeterminado, se usan a menudo otros como el punto y coma (;) o la tabulación (\t)—este último se conoce como TSV (*Tab Separated Values*). |
| Encabezados | Generalmente, la primera línea contiene los nombres de las columnas o campos, facilitando el mapeo de los datos durante su importación. |

```
Name,Age,Occupation
Alice,30,Engineer
Bob,25,Data Scientist
Charlie,35,Teacher
```

El CSV es el formato más común para la importación y exportación de datos entre diferentes aplicaciones, como bases de datos, sistemas ERP y hojas de cálculo (Excel, Google Sheets). A pesar de su uso extendido, el formato carece de un estándar estricto unificado (aunque existe el intento **RFC 4180**). Esta falta de estandarización da lugar a **sutiles diferencias** en la forma en que las aplicaciones producen y consumen los archivos, lo que obliga a los programas de procesamiento (como el módulo csv de Python) a ser flexibles.

### Procesamiento de Archivos CSV con Python

Python ofrece dos herramientas fundamentales para el manejo de archivos CSV: la librería estándar **csv** para control de bajo nivel y **pandas** para análisis de datos de alto nivel.

### El Módulo Estándar

El módulo integrado csv está diseñado para abstraer las complejidades del formato (como los diferentes delimitadores y las reglas de quoting), permitiendo al programador centrarse solo en la lógica de los datos.

Dialectos (Dialect)

Debido a la variación del formato, Python utiliza el concepto de **Dialectos** para agrupar parámetros específicos de formato. Un dialecto define:

- **delimiter**: El delimitador usado.
- **quotechar**: El carácter de comilla de campo.
- **skipinitialspace**: Si se ignoran los espacios después del delimitador.
- **lineterminator**: El carácter utilizado para terminar las líneas.

El dialecto por defecto es 'excel', pero se pueden registrar dialectos personalizados o utilizar predefinidos como 'unix_dialect' o 'excel-tab'

**Clases de Lectura y Escritura**

El módulo proporciona clases específicas para trabajar con los datos:.

| Objeto | Tipo de Entrada/Salida | Uso Común en Informes |
|---|---|---|
| **csv.reader()** | Lee filas como listas de cadenas. | Ideal para iterar rápidamente sobre archivos grandes donde el orden es fijo. |
| **csv.writer()** | Escribe filas desde listas a un archivo. | Utilizado para escribir secuencias de datos sin necesidad de encabezados. |
| **csv.DictReader()** | Lee filas como diccionarios (dict), usando los encabezados como claves. | Muy recomendado. Permite acceder a los campos por su nombre (ej: row['Nombre']), mejorando la legibilidad. |
| **csv.DictWriter()** | Escribe filas desde diccionarios, requiriendo una secuencia de nombres de campo (fieldnames) para definir el orden. | Ideal para generar nuevos archivos CSV, incluyendo la función writeheader() para escribir la primera fila con los encabezados. |

**Lectura básica de CSV**

```
import csv

# Open a CSV file
with open('example.csv', 'r') as file:
    reader = csv.reader(file)

    # Iterate over the rows
    for row in reader:
        print(row)
```

**Manejo de encabezados**
La mayoría de los archivos CSV vienen con encabezados en la primera fila, como los nombres de las columnas. Si no necesita estos encabezados, simplemente puede omitir la primera fila al iterar:

```
import csv

with open('example.csv', 'r') as file:
    reader = csv.reader(file)

    # Skip header
    next(reader)

    for row in reader:
        print(row)
```

**DictReader**

Si su archivo CSV tiene encabezados, csv.DictReader() es otra opción fantástica que lee cada fila como un diccionario, donde las claves son los nombres de las columnas:

```
import csv

with open('example.csv', 'r') as file:
    reader = csv.DictReader(file)

    for row in reader:
        print(row)
```

**Escritura básica de CSV**

La función writer.writerows() toma una lista de listas y las escribe en el archivo CSV, donde cada lista interna representa una fila de datos.

```
import csv

# Data to be written
data = [
    ['Name', 'Age', 'Occupation'],
    ['Alice', 30, 'Engineer'],
    ['Bob', 25, 'Data Scientist'],
    ['Charlie', 35, 'Teacher']
]

# Open a file in write mode
with open('output.csv', 'w', newline='') as file:
    writer = csv.writer(file)

    # Write data to the file
    writer.writerows(data)
```

**DictWriter**

Este método puede ser especialmente útil cuando desea especificar los nombres de sus columnas explícitamente.

```
import csv

# Data as list of dictionaries
data = [
    {'Name': 'Alice', 'Age': 30, 'Occupation': 'Engineer'},
    {'Name': 'Bob', 'Age': 25, 'Occupation': 'Data Scientist'},
    {'Name': 'Charlie', 'Age': 35, 'Occupation': 'Teacher'}
]

# Open file for writing
with open('output.csv', 'w', newline='') as file:
    fieldnames = ['Name', 'Age', 'Occupation']
    writer = csv.DictWriter(file, fieldnames=fieldnames)

    # Write the header
    writer.writeheader()

    # Write the data
    writer.writerows(data)
```

Bibliografia

Listas

https://curso-python.org/list.html

https://docs.python.org/es/3.13/tutorial/datastructures.html

Diccionarios

https://www.freecodecamp.org/espanol/news/compresion-de-diccionario-en-python-explicado-con-ejemplos/

https://docs.python.org/es/3.13/tutorial/datastructures.html

https://ellibrodepython.com/diccionarios-en-python

Funciones

https://ellibrodepython.com/funciones-en-python

https://j2logo.com/python/tutorial/funciones-en-python/#function-params

https://elpythonista.com/funciones-en-python-guia-completa-2025-sintaxis-parametros-y-ejemplos

Condicionales

https://ellibrodepython.com/if-python

https://docs.python.org/es/3.13/reference/compound_stmts.html

https://www.mclibre.org/consultar/python/lecciones/python-if-else.html

https://www.programaenpython.com/fundamentos/sentencias-condicionales-en-python/#google_vignette

Ordenamiento

https://developers.google.com/edu/python/sorting?hl=es-419

https://ellibrodepython.com/bubble-sort

https://4geeks.com/es/lesson/algoritmos-de-ordenamiento-y-busqueda-en-python

https://eiposgrados.com/blog-python/tipos-de-algoritmos-de-ordenacion-en-python/

Estadisticas Basicas

https://docs.python.org/3/library/statistics.html

https://interactivechaos.com/es/manual/tutorial-de-python/la-libreria-statistics

https://rico-schmidt.name/pymotw-3/statistics/index.html

https://oregoom.com/python/estadisticas/

Archivos CSV

https://dev.to/devasservice/guide-to-pythons-csv-module-32ie

https://docs.python.org/3/library/csv.html

https://www.geeknetic.es/Archivo-CSV/que-es-y-para-que-sirve