Trabajo Integrador de Programación

Alumnos

Ivanna Ávila y Tarica Lola

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Programación

Docente Titular

Alberto Cortez

Docente Tutor

Miguel Barrera

11 de Noviembre de 2025

ÍNDICE DEL DOCUMENTO

Marco teórico

Listas en python

Introducción

En el desarrollo de software, el manejo eficiente de datos es clave para construir soluciones funcionales y escalables. Python, como lenguaje de programación moderno y versátil, ofrece estructuras de datos poderosas que facilitan esta tarea. Entre ellas, las listas destacan por su simplicidad, flexibilidad y amplio rango de aplicaciones.

Las listas permiten almacenar colecciones de elementos de forma ordenada y mutable, lo que las convierte en una herramienta fundamental tanto en proyectos educativos como en entornos profesionales. Desde el procesamiento de datos hasta la automatización de tareas, su uso se extiende a múltiples áreas como la inteligencia artificial, el desarrollo web, la gestión de inventarios y la creación de videojuegos.

Este marco teórico presenta una visión integral sobre las listas en Python, combinando definiciones, ejemplos prácticos, métodos clave y aplicaciones reales. El objetivo es ofrecer una base sólida para comprender su funcionamiento y potenciar su uso en proyectos de programación.

¿Qué es una lista?

Una lista en Python es una estructura de datos que permite almacenar múltiples elementos en una sola variable. Se define utilizando corchetes [ ] y puede contener distintos tipos de datos, como números, cadenas de texto, booleanos o incluso otras listas.

Ejemplos:

- Lista de números: [1, 2, 3, 4, 5]
- Lista de cadenas: ["Ana", "Luis", "Pedro"]
- Lista mixta: [1, "Hola", 3.14, True]
- Lista vacía: [ ]

Tipos de datos en listas

Las listas pueden contener:

- Números enteros y decimales
- Cadenas de texto
- Booleanos (True o False)
- Otras listas (listas anidadas)

Ejemplo:

```
[42, "texto", False, [1, 2, 3]]
```

Acceso a elementos

Los elementos se acceden por índice:

- Índices positivos:

nombres[0] sobre ["Ana", "Luis", "Pedro"] devuelve "Ana"

- Índices negativos:

nombres[-1] devuelve "Pedro"

- Listas anidadas:

anidada[0][1] sobre [[1, 2, 3], [4, 5, 6]] devuelve 2

Creación con range()

- list(range(5)) → [0, 1, 2, 3, 4]
- list(range(2, 10)) → [2, 3, 4, 5, 6, 7, 8, 9]
- list(range(2, 10, 2)) → [2, 4, 6, 8]

Slicing de listas

Permite extraer subconjuntos:

- letras[1:4] sobre ["a", "b", "c", "d", "e"] → ["b", "c", "d"]
- letras[:3] → ["a", "b", "c"]
- letras[2:] → ["c", "d", "e"]
- letras[-3:-1] → ["c", "d"]

Métodos principales de listas

append(x): agrega x al final

[1, 2, 3].append(4) → [1, 2, 3, 4]

extend(iterable): agrega todos los elementos

[1, 2].extend([3, 4]) → [1, 2, 3, 4]

insert(i, x): inserta x en la posición i

["a", "b", "d"].insert(2, "c") → ["a", "b", "c", "d"]

remove(x): elimina el primer x

[1, 2, 3, 2].remove(2) → [1, 3, 2]

pop(i): elimina y retorna el elemento en i

[10, 20, 30].pop(1) → retorna 20, lista queda [10, 30]

clear(): elimina todos los elementos

[1, 2, 3].clear() → [ ]

index(x): retorna el índice del primer x

["rojo", "verde", "azul"].index("verde") → 1

count(x): cuenta cuántas veces aparece x

[1, 2, 2, 3].count(2) → 2

sort(): ordena la lista

[3, 1, 4, 2].sort() → [1, 2, 3, 4]

reverse(): invierte el orden

[1, 2, 3].reverse() → [3, 2, 1]

copy(): copia la lista original

[1, 2, 3]; copia = original.copy() → [1, 2, 3]

<u>Conversión con split() y modificación</u>

- "Hola mundo Python".split() → ["Hola", "mundo", "Python"]
- "manzana,banana,pera".split(",") → ["manzana", "banana", "pera"]
- [1, 2, 3].append(4) → [1, 2, 3, 4]
- [1, 2, 3].remove(2) → [1, 3]

Concatenación y repetición

- [1, 2, 3] + [4, 5, 6] → [1, 2, 3, 4, 5, 6]
- [0] * 4 → [0, 0, 0, 0]

Uso del operador in

- 3 in [1, 2, 3, 4] → True
- for fruta in ["manzana", "banana", "naranja"]: recorre cada elemento

Aplicaciones reales

- Inventarios: productos = ["notebook", "mouse", "monitor"]
- Datos: temperaturas = [22.5, 23.1, 21.8]
- Automatización: tareas = ["descargar", "procesar", "informar"]
- IA: predicciones = [0, 1, 1, 0]
- Videojuegos: enemigos = ["zombi", "dragón"]
- Interfaces: menu = ["Inicio", "Perfil", "Salir"]

Ejemplos prácticos

Crear una lista de frutas favoritas, añadir una fruta con append() y eliminar otra con remove()

frutas = ["manzana", "plátano", "fresa", "mango","piña"] frutas.append("naranja")→ ["manzana", "plátano", "fresa", "mango", "piña", "naranja"]

frutas.remove("plátano") → ["manzana", "fresa", "mango", "piña", "naranja"]

Usar slicing para obtener los primeros tres elementos, los últimos dos y los elementos en posiciones pares de una lista.

numeros = [10, 20, 30, 40, 50, 60, 70, 80, 90]

Primeros tres: numeros[:3] → [10, 20, 30]

Últimos dos: numeros[-2:] → [80, 90]

Posiciones pares:

numeros[::2] → [10, 30, 50, 70, 90]

Convertir una cadena de lenguajes de programación en lista con split() y añadir un nuevo lenguaje.

texto="Python,Java,C++,JavaScript,PHP" lenguajes = texto.split(",") → ["Python", "Java", "C++", "JavaScript", "PHP"] lenguajes.append("Ruby") → ["Python", "Java", "C++", "JavaScript", "PHP", "Ruby"]

Conclusión

Las listas en Python representan una herramienta esencial para el desarrollo de soluciones dinámicas, eficientes y adaptables. Su capacidad para almacenar y manipular datos de forma ordenada y mutable las convierte en una estructura versátil, presente en proyectos de diversa complejidad, desde scripts simples hasta sistemas avanzados de inteligencia artificial. A lo largo de este marco teórico se han abordado sus características fundamentales, los métodos más utilizados, técnicas de acceso y modificación, así como ejemplos prácticos que ilustran su aplicación en contextos reales. Comprender el funcionamiento de las listas no solo fortalece la lógica de programación, sino que también habilita al desarrollador para construir algoritmos más robustos y optimizar el tratamiento de datos en sus proyectos. Dominar esta estructura es un paso clave en el camino hacia una programación más clara, modular y profesional.

Diccionarios en Python

Introducción

En el ámbito de la programación, las estructuras de datos permiten organizar la información de manera lógica y eficiente. Python ofrece diversas estructuras, entre ellas los diccionarios, que destacan por su capacidad de asociar valores a claves únicas. Esta característica los convierte en herramientas esenciales para modelar datos estructurados, realizar búsquedas rápidas y representar relaciones jerárquicas.

Definición y características

Un diccionario es una colección mutable de pares clave-valor, donde cada clave debe ser única e inmutable (por ejemplo, cadenas, números o tuplas),

y cada valor puede ser de cualquier tipo. A diferencia de las listas o tuplas, los diccionarios no se acceden por posición, sino por clave.

Características principales:

- Acceso por clave: se accede al valor mediante su clave, por ejemplo estudiante["nombre"].
- Claves únicas e inmutables: no se permiten claves duplicadas.
- Valores flexibles: pueden ser números, listas, otros diccionarios, etc.
- Mutabilidad: se pueden modificar, agregar o eliminar pares clave-valor.
- Desde Python 3.7, los diccionarios mantienen el orden de inserción.

Ejemplos

Crear un diccionario

```
Estudiante =
{
  "nombre": "Lucía",
  "edad": 20,
  "carrera": "Informática"
}
```

Acceder a un valor

```
estudiante["nombre"] devuelve "Lucía"
```

Agregar un nuevo dato

```
estudiante["promedio"] = 8.75
```

Modificar un valor existente

```
estudiante["edad"] = 21
```

Eliminar un dato

```
estudiante.pop("carrera")
```

Métodos comunes

get(clave, valorpordefecto): acceso seguro sin generar error si la clave no existe.
update(): agrega o modifica múltiples pares clave-valor.
setdefault(): asigna un valor si la clave no existe.
pop(), popitem(): eliminan elementos y retornan su valor.
keys(), values(), items(): devuelven vistas de claves, valores y pares respectivamente.

Recorrer un diccionario

for clave, valor in estudiante.items(): imprimir clave y valor

Diccionarios anidados y listas de diccionarios

Estas variantes permiten representar estructuras más complejas y jerárquicas.

Ejemplo de diccionario anidado

```
escuela =
{
  "1A": {
          "tutor": "Ana",
          "alumnos": ["Luis", "María"]
  }
}
```

Ejemplo de lista de diccionarios

```
usuarios =
[
  {"id": 1, "nombre": "Pablo"},
  {"id": 2, "nombre": "Sofía"}
]
```

Comprensiones de diccionarios

Las comprensiones permiten crear diccionarios de forma compacta y eficiente.

Ejemplo

```
cuadrados =
{
  0: 0,
  1: 1,
  2: 4,
  3: 9,
  4: 16
}
```

Esto se genera con una expresión como:
{x: x2 para x en el rango de 0 a 4}

Aplicaciones prácticas

Los diccionarios se utilizan en múltiples contextos:

- Modelado de datos: usuarios, productos, configuraciones.
- Representación de objetos con atributos.

- Análisis de texto con collections.Counter.

Ejemplo con Counter

```
frutas = ["manzana", "naranja", "manzana", "plátano"]
conteo = Counter(frutas)
Resultado: manzana aparece 2 veces, naranja 1 vez, plátano 1 vez
```

Buenas prácticas

- Usar get() para evitar errores por claves inexistentes.
- Documentar el uso de claves y estructura para facilitar el mantenimiento.
- No modificar el diccionario mientras se recorre.
- Considerar el rendimiento en estructuras grandes.

Errores comunes

- Acceder a claves inexistentes sin validación previa.
- Modificar el diccionario durante una iteración.
- Confundir la inmutabilidad de las claves con la de los valores.

Conclusión

Los diccionarios en Python son herramientas poderosas para organizar y acceder a datos de forma eficiente. Su flexibilidad, velocidad y capacidad de anidación los convierten en una opción ideal para múltiples aplicaciones, desde el modelado de datos hasta el análisis de información. Comprender su funcionamiento y aplicarlos correctamente es fundamental para escribir código limpio, estructurado y funcional.

Las estructuras de datos permiten organizar la información de manera lógica y eficiente. Python ofrece diversas estructuras, entre ellas los diccionarios, que destacan por su capacidad de asociar valores a claves únicas. Esta característica los convierte en herramientas esenciales para modelar datos estructurados, realizar búsquedas rápidas y representar relaciones jerárquicas.

Funciones en Python

Introducción

Las funciones en Python son bloques fundamentales para la construcción de programas eficientes, legibles y reutilizables. Permiten encapsular lógica

específica, facilitando la resolución de problemas complejos mediante soluciones modulares. Este marco teórico explora en profundidad el concepto de funciones en Python, sus tipos, ventajas, estructura, y buenas prácticas, complementando la información del documento académico con la documentación oficial del lenguaje.

¿Qué es una función?

Una función en Python es un bloque de código reutilizable que realiza una tarea específica. Actúa como una 'caja negra' que recibe datos (argumentos), los procesa internamente y devuelve un resultado. Esta estructura permite dividir problemas complejos en partes más manejables.

Elementos de una función

Argumentos: Entrada

Datos que se suministran a la función, como números, cadenas de texto o estructuras complejas.

Cuerpo: Proceso

Conjunto de instrucciones que procesan la información recibida.

Retorno: Salida

Resultado final obtenido después del procesamiento.

Ventajas de usar funciones

- Reutilización de código: permite escribir una vez y utilizar múltiples veces.
- Mantenimiento simplificado: modificar una función actualiza todas sus instancias.
- Abstracción: oculta la complejidad interna, facilitando el uso.
- Legibilidad mejorada: el código es más organizado y comprensible.

Tipos de funciones

Definición: Funciones integradas (built-in)

Son funciones que ya vienen incluidas en Python, como print(), len(), input().

Definición: Funciones definidas por el usuario

Son creadas por el programador para resolver tareas específicas, utilizando la palabra clave 'def'.

Ejemplo de función

Función que suma dos número

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

x = 5
y = 3
suma = sumar(x, y)
print("La suma es:", suma)
```

Parámetros en funciones

Definición: Por valor

La función recibe una copia del valor original. Los cambios dentro de la función no afectan la variable externa. Este comportamiento es típico en tipos inmutables como int, float y str.

Definición: Por referencia

La función recibe una referencia al objeto original, compartiendo el mismo espacio en memoria. Este comportamiento es común en tipos mutables como list, dict y set.

Composición de funciones

La composición de funciones ocurre cuando el resultado de una función se utiliza como argumento de otra. Esto permite escribir código más compacto y expresivo, aunque puede ser útil usar variables intermedias para mejorar la claridad.

Closures

Un closure en Python es una función interna que recuerda el estado de las variables del entorno donde fue creada, incluso después de que ese entorno haya terminado su ejecución. Para que exista un closure deben cumplirse tres condiciones:

- Tener una función definida dentro de otra.
- La función interna utiliza variables de la función externa.
- La función externa retorna la función interna.

Ejemplo de Closure

```
def crear_contador():
        contador = 0
        def incrementar():
        nonlocal contador
        contador += 1
        return contador
        return incrementar

 mi_contador = crear_contador()
 print(mi_contador())  # 1
 print(mi_contador())  # 2
```

Buenas prácticas en el uso de funciones

- Usar nombres descriptivos que indiquen claramente el propósito de la función.
- Cada función debe tener una única responsabilidad.
- Documentar las funciones utilizando docstrings.
- Evitar código duplicado extrayendo bloques repetidos en funciones.
- Probar funciones con casos límite para asegurar robustez.

Ámbitos de variables

Definición: Ámbito local

Variables declaradas dentro de una función solo existen en ese contexto.

Definición: Ámbito global

Variables creadas fuera de cualquier función son accesibles desde todo el archivo.

Modificación de variables globales

Para modificar variables globales dentro de una función se utiliza la palabra clave 'global'. Aunque es posible, se recomienda usarla con precaución para evitar problemas de mantenimiento.

Ejemplos

Calcular el factorial de un número

```
def factorial(n):
    if n == 0 or n == 1:
    return 1
    else:
    return n * factorial(n - 1)

# Ejemplo de uso
print(factorial(5))  # Salida: 120
```

La función utiliza recursión para calcular el producto de todos los números desde n hasta 1. Si n es 0 o 1, retorna 1 como caso base.

Verificar si un número es primo

```
def es_primo(n):
    if n < 2:
    return False
    for i in range(2, int(n**0.5) + 1):
    if n % i == 0:
    return False
    return True

# Ejemplo de uso
print(es_primo(7))  # Salida: True
```

La función verifica si el número tiene divisores distintos de 1 y de sí mismo. Si encuentra alguno, retorna False. Si no, retorna True.

Sumar los elementos de una lista

```
def sumar_lista(lista):
    return sum(lista)
```

```
# Ejemplo de uso
print(sumar_lista([1, 2, 3, 4]))  # Salida: 10
```

La función utiliza la función integrada sum() para calcular la suma total de los elementos de la lista.

Conclusión

Las funciones en Python son herramientas poderosas que permiten estructurar el código de manera eficiente, clara y reutilizable. Comprender su funcionamiento, tipos, y aplicar buenas prácticas es esencial para desarrollar programas robustos y mantenibles. El uso adecuado de funciones facilita la resolución de problemas complejos y mejora la calidad del software desarrollado.

Estructuras Condicionales

Introducción

Las estructuras condicionales son fundamentales en la programación, ya que permiten que un programa tome decisiones basadas en condiciones específicas. En Python, estas estructuras son esenciales para crear aplicaciones dinámicas, interactivas y capaces de responder a diferentes escenarios.

¿Qué son las estructuras condicionales?

Las estructuras condicionales son mecanismos de control que permiten a los algoritmos tomar decisiones basadas en si se cumple o no una condición. Representan el pensamiento lógico detrás de los programas informáticos y son esenciales para crear software dinámico y receptivo.

Clasificación de las estructuras condicionales en Python

Condicional simple (if)

Ejecuta un bloque de código solo si una condición es verdadera.

Ejemplo

```
if temperatura > 30:
        print("¡Hace calor afuera!")
        print("Recuerda mantenerte hidratado")
```

Condicional compuesto (if-else)

Proporciona una ruta alternativa cuando la condición es falsa.

Ejemplo

```
if edad >= 18:
        print("Eres un adulto")
 else:
        print("Eres menor de edad")
```

Condicional múltiple (if-elif-else)

Permite manejar múltiples condiciones y rutas posibles.

Ejemplo

```
if score >= 90:
        print("Calificación: A")
 elif score >= 80:
        print("Calificación: B")
 else:
        print("Calificación: F")
```

Condicionales anidados

Permiten estructuras de decisión más complejas colocando condicionales dentro de otros.

Ejemplo

```
if usuario == "admin":
        if contraseña == "1234":
        print("Acceso concedido")
        else:
        print("Contraseña incorrecta")
 else:
        print("Usuario desconocido")
```

Operadores Relacionales y Lógicos

Operadores Relacionales

== : Igual a
!= : Distinto de
> : Mayor que
< : Menor que
>= : Mayor o igual que
<= : Menor o igual que

Operadores Lógicos

and : Verdadero si ambas condiciones son verdaderas
or : Verdadero si al menos una condición es verdadera
not : Invierte el valor de verdad de una condición

Ejemplo con 'and':

```
if edad >= 18 and tiene_id:
        print("Acceso permitido")
```

Ejemplo con 'or':

```
if es_fin_de_semana or es_feriado:
        print("Día libre")
```

Ejemplo con 'not':

```
if not es_hora_laboral:
        print("No estás trabajando")
```

Errores comunes en condicionales

- Usar '=' en lugar de '==' para comparar valores.
- Olvidar los dos puntos ':' al final de la condición.
- Indentación incorrecta que causa errores de sintaxis.

Ejemplo práctico: Calculadora de IMC

```
altura = float(input("Ingrese su altura en metros: "))
peso = float(input("Ingrese su peso en kilogramos: "))
imc = peso / (altura ** 2)
print(f"Su IMC es: {imc:.2f}")
if imc < 18.5:
        print("Bajo peso")
elif imc < 25:
        print("Peso normal")
elif imc < 30:
```

```
        print("Sobrepeso")
else:
        print("Obesidad")
```

El poder de las estructuras condicionales

Las estructuras condicionales permiten que el código responda de manera diferente según las entradas, modelando decisiones del mundo real, manejando errores y casos excepcionales, e implementando lógica de negocios compleja.

Ejemplos

Verificación de temperatura

```
temperature = float(input("Ingrese la temperatura en °C: "))
if temperature > 30:
        print("¡Hace calor afuera!")
print("Fin de la verificación.")
```

Se utiliza una estructura condicional simple (`if`) para verificar si la temperatura supera los 30°C. Si es así, se muestra un mensaje.

Verificación de acceso

```
edad = int(input("Ingrese su edad: "))
tiene_id = input("¿Tiene identificación? (s/n): ") == "s"
if edad >= 18 and tiene_id:
        print("Acceso permitido")
else:
        print("Acceso denegado")
```

Se utiliza un condicional compuesto (`if-else`) con el operador lógico `and` para verificar dos condiciones simultáneamente.

Clasificación de calificaciones

```
score = int(input("Ingrese la calificación: "))
if score >= 90:
        print("Calificación: A")
elif score >= 80:
        print("Calificación: B")
elif score >= 70:
        print("Calificación: C")
```

```
elif score >= 60:
        print("Calificación: D")
else:
        print("Calificación: F")
```

Se utiliza una estructura condicional múltiple (`if-elif-else`) para evaluar rangos de calificaciones. Se debe prestar atención a la indentación y al uso correcto de los dos puntos (`:`).

Conclusión

Dominar las estructuras condicionales en Python es esencial para cualquier programador. Estas herramientas permiten crear programas inteligentes, adaptables y eficientes, capaces de tomar decisiones y responder a diferentes situaciones de forma lógica y estructurada.

Ordenamiento

El ordenamiento es el proceso de organizar los elementos de una colección (listas) según un criterio:

- Numérico (menor a mayor)
- Alfabético (A - Z)
- Objetos, osea por un atributo (ejemplo: ordenar empleados por edad).

En la programación esto es clave para facilitar la búsqueda, organización y análisis de datos.

Python tiene varios métodos y algoritmos que nos ayudan para el ordenamiento.

Métodos

List.sort(): Ordena la lista en ella misma.

Sintaxis

nombre_lista.sort()

Ejemplo

Tenemos un conjunto de números que queremos ordenar de menor a mayor.

numeros = [2,3,6,1,9]
numeros.sort()
print(numeros)

[1,2,3,6,9]

Sorted(): Ordena la lista y lo agrega a otro iterable
En este caso nuestra variable original no cambia

Sintaxis

nueva_var = sorted(lista)

Ejemplo
Tenemos un conjunto de números que queremos ordenar de menor a mayor y asignarlos a una nueva variable

numeros = [2,3,6,1,9]
ordenada = sorted(numeros)
print(numeros)

[1,2,3,6,9]

Min y max: encuentran el mínimo/máximo de una o más iterables

Sintaxis

min(lista)
max(lista)

Ejemplo
Buscar el número menor y mayor de una lista

numeros = [2,3,6,1,9]
print(min(numeros)) # 1
print(max(numeros)) # 9

Heapq: obtiene el más pequeño/grande sin ordenar la lista
Se puede traer una cantidad que uno quiera, siendo bastante útil para cuando querés saber los primeros más chicos o más grande, etc.

Sintaxis

heaplq.nsmallest(cant, lista)
heaplq.nlargest(cant, lista)

Ejemplo
Traer los dos número más grandes y más chicos de la lista

```
import heapq
numeros = [5, 2, 9, 1, 7]
print(heapq.nsmallest(2, numeros))  # [1, 2]
print(heapq.nlargest(2, numeros))   # [9, 7]
```

También tenemos dos parámetros para modificar la forma en que ordenan ciertos métodos: reverse y key

Reverse: Ordena en de forma descendente

Ejemplo
Tenemos un conjunto de números que queremos ordenar de menor a mayor.

```
numeros = [2,3,6,1,9]
numeros.sort()
print(numeros)
```

```
[9,6,3,2,1]
```

Key: decide cómo comparar partiendo de un atributo

Ejemplo
Tenemos una lista de palabras y queremos ordenar de menor a mayor cantidad de caracteres (longitud)

```
palabras = ["sol", "estrella", "luz"]
print(sorted(palabras, key=len))
# ['sol', 'luz', 'estrella']
```

Métodos que utilizan parámetros		
Método	**key**	**reverse**
list.sort()	Sí	Sí
sorted()	Sí	Sí

min()	Sí	No
max()	Sí	No
heapq.nsmallest()	Sí	No
heapq.nlargest()	Sí	No

Algoritmos

Aunque son útiles no suelen usarse ya que varios métodos hacen lo mismo con más facilidad. Podrían ser reemplazados por list.sort o sorted

Bubble sort: Compara pares de elementos y intercambia si están de forma incorrecta

Ejemplo

```
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]

numeros = [5, 2, 9, 1, 7]
bubble_sort(numeros)
print(numeros)
```

```
 [1, 2, 5, 7, 9]
```

Inserción sort: inserta cada elemento en la posición correcta dentro de la parte ya ordenada

Ejemplo

```
def insertion_sort(lista):
    for i in range(1, len(lista)):
        clave = lista[i]
        j = i - 1
        while j >= 0 and clave < lista[j]:
            lista[j + 1] = lista[j]
            j -= 1
        lista[j + 1] = clave
```

```
numeros = [5, 2, 9, 1, 7]
insertion_sort(numeros)
print(numeros)

[1, 2, 5, 7, 9]
```

Selection sort: busca el mínimo en cada pasada y lo coloca en la posición correcta

Ejemplo

```
def selection_sort(lista):
    n = len(lista)
    for i in range(n):
        minimo = i
        for j in range(i + 1, n):
            if lista[j] < lista[minimo]:
                minimo = j
        lista[i], lista[minimo] = lista[minimo], lista[i]

numeros = [5, 2, 9, 1, 7]
selection_sort(numeros)
print(numeros)
# [1, 2, 5, 7, 9]
```

Merge sort: divide las listas en mitades y las ordena recursivamente para después combinarlas

Ejemplo

```
def merge_sort(lista):
    if len(lista) > 1:
        medio = len(lista) // 2
        izquierda = lista[:medio]
        derecha = lista[medio:]

        merge_sort(izquierda)
        merge_sort(derecha)

        i = j = k = 0
        while i < len(izquierda) and j < len(derecha):
            if izquierda[i] < derecha[j]:
                lista[k] = izquierda[i]
                i += 1
            else:
                lista[k] = derecha[j]
                j += 1
```

```
            k += 1

        while i < len(izquierda):
            lista[k] = izquierda[i]
            i += 1
            k += 1

        while j < len(derecha):
            lista[k] = derecha[j]
            j += 1
            k += 1

numeros = [5, 2, 9, 1, 7]
merge_sort(numeros)
print(numeros)
 [1, 2, 5, 7, 9]
```

Quick sort: escoge un pivote (el cual comparar), divide la lista en menores y mayores y los ordena.

Ejemplo

```
def quick_sort(lista):
    if len(lista) <= 1:
        return lista
    else:
        pivote = lista[0]
        menores = [x for x in lista[1:] if x <= pivote]
        mayores = [x for x in lista[1:] if x > pivote]
        return quick_sort(menores) + [pivote] + quick_sort(mayores)

numeros = [5, 2, 9, 1, 7]
print(quick_sort(numeros))
# [1, 2, 5, 7, 9]
```

Estadísticas básica

La estadística es la rama de las matemáticas que escoge, organiza, analiza e interpreta los datos para entender información y tomar decisiones. Dentro de esta existen las medidas estadísticas que ayudan a resumir y describir un conjunto de datos.

Medias estadisticas

Tendencia central – el centro de los datos

Media: promedio
Suma todos los números y divide por la cantidad de estos.

Sintaxis

import statistics
statitistics.mean(numeros)

Ejemplo

```
import statistics
print(statistics.mean([10, 20, 30]))
```

20

Mediana: número que queda en el medio al ordenar los datos de menor a mayor

Sintaxis

import statistics
statitistics.median(numeros)

Ejemplo

```
import statistics
print(statistics.median([10, 30, 20]))
```

20

Moda: el número que más aparece

Sintaxis

import statistics
statistics.mode(numeros)

Ejemplo

```
datos = [10, 20, 20, 30]
import statistics
print(statistics.mode(datos))  # 20
```

Dispersión: diferencias entre datos, no solo el centro si no sus cercanos y lejanos

Rango: diferencia entre el valor más alto y más bajo

Sintaxis

max(lista) - min(lista)

Ejemplo

edades = [19, 15]
print(max(edades) - min(edades))

19-15 = 4

Varianza: mide cuánto se alejan los datos del promedio, al cuadrado

Sintaxis

statistics.variance(lista)

Desviación: que tan alejado está un numero de la media

Sintaxis

statistics.stdev(lista)

Percentiles y Cuartiles: dividen en grupos según el promedio

Sintaxis

statistics.quantiles(lista, promedio)

Ejemplo

Separar al grupo de clase en cuartiles:

- Primer cuartil (Q1): el 25 % de alumnos con notas más bajas.

- Segundo cuartil (Q2): mediana, 50 % de alumnos por debajo y 50 % por encima.

- Tercer cuartil (Q3): 25 % de alumnos con notas más altas.

print(statistics.quantiles(notas, n=4))

[5.25, 7, 8] → Q1=5.25, Q2=7, Q3=8

Archivos CSV

El CSV (Comma Separated Values) es un formato de texto plano para almacenar datos en forma de tabla. Cada fila representa un registro y cada columna un valor, separados por un carácter especial (por ejemplo, “,” o “;”). Este formato se usa para el intercambio de datos entre programas y Python lo maneja de forma nativa.

Ejemplo de archivo CSV

columna1, columna2, columna3
dato1, dato2, dato3
dato4, dato5, dato6

Codificación (Encoding)

Los archivos de texto pueden estar codificados en distintos formatos:

- UTF-8: estándar actual, soporta tildes y ñ.
- latin-1 o ISO-8859-1: usado por sistemas antiguos.

En Python:

open("archivo.csv", "r", encoding="utf-8")

Uso de with open(...) (context manager)

Usar with es la forma recomendada de manejar archivos porque:

- Cierra el archivo automáticamente al terminar.
- Evita errores o fugas de recursos si el programa falla.

```
with open("archivo.csv", "r") as f:
   contenido = f.read()
# El archivo ya está cerrado aquí
```

También se puede usar junto con el módulo os:

```
import os
if os.path.exists("datos.csv"):
   os.remove("datos.csv")
```

Saltos de línea

En Windows las líneas terminan con \r\n, mientras que en Linux/Mac solo con \n. Python maneja esto automáticamente si se abre el archivo con newline="".

```
open("archivo.csv", "r", newline="", encoding="utf-8")
```

Librería csv y lectura básica

```
import csv

with open("datos.csv", newline="", encoding="utf-8") as f:
   lector = csv.reader(f)
   for fila in lector:
      print(fila)
```

Lectura como diccionarios

Si el CSV tiene encabezados, se puede leer como diccionarios:

```
with open("datos.csv", newline="", encoding="utf-8") as f:
   lector = csv.DictReader(f)
   for fila in lector:
      print(fila["nombre"], fila["edad"])
```

Creación e importación de archivos CSV

```
import csv

with open("nuevo.csv", "w", newline="", encoding="utf-8") as f:
   escritor = csv.writer(f)
   escritor.writerow(["nombre", "edad", "ciudad"])
   escritor.writerow(["Lucía", 28, "Córdoba"])
```

- El modo "w" crea o sobrescribe el archivo.
- El modo "a" agrega contenido sin borrar lo existente.

Modos de edición

- "r" → leer
- "w" → escribir (borra lo anterior)
- "a" → agregar
- "r+" → leer y escribir sin borrar
- "w+" → crear o sobrescribir y luego leer

Funciones útiles de strings

Estas funciones se usan mucho al procesar texto línea por línea.

strip(): elimina espacios o saltos de línea al principio y al final.
Ejemplo

" hola \n".strip() → "hola"

split(","): divide un texto en una lista según el separador indicado.
Ejemplo

"Ana,25".split(",") → ['Ana', '25']

casefold() o lower(): convierte a minúsculas para evitar diferencias de mayúsculas y minúsculas.
Ejemplo

"HOLA".casefold() → "hola"

capitalize() o title(): cambia el formato de mayúsculas de forma controlada.

Case sensitive: en Python se distinguen mayúsculas y minúsculas.
Ejemplo

"Hola" == "hola" → False

Lectura línea por línea

```
with open("datos.csv", "r", encoding="utf-8") as f:
    linea = f.readline()   # Lee una sola línea
    todas = f.readlines()  # Lee todas las líneas como lista
```

read(): lee todo el archivo como una sola cadena.
readline(): lee una sola línea.
readlines(): devuelve una lista con todas las líneas.

También se puede usar for linea in f: para recorrer línea por línea sin cargar todo el archivo en memoria.

Reescribir, editar y eliminar líneas

Reescribir

```
with open("datos.csv", "w", encoding="utf-8") as f:
    f.writelines(["nombre,edad\n", "Ana,25\n", "Juan,30\n"])
```

Editar una línea específica:

```
with open("datos.csv", "r", encoding="utf-8") as f:
    lineas = f.readlines()

lineas[1] = "Ana,26\n"

with open("datos.csv", "w", encoding="utf-8") as f:
    f.writelines(lineas)
```

Eliminar una línea:

```
nuevas = [l for l in lineas if not l.startswith("Juan")]
with open("datos.csv", "w", encoding="utf-8") as f:
    f.writelines(nuevas)
```

Métodos write, writelines, readline, readlines

write("texto"): escribe una sola cadena de texto.

writelines(lista): escribe varias líneas (no agrega saltos de línea automáticamente).

readline(): lee una sola línea.
readlines(): devuelve todas las líneas en forma de lista.

Flujo de operaciones principales del sistema

Link:
https://www.figma.com/board/GBix1WviTGDZTFLrvtTOE5/Flujo-sistema?node-id=0-1&t=TE0KsRx7GouHywQ2-1

Capturas de ejecución de ejemplos

Func 1. Agregar país

- Ingreso correcto

```
===== MENÚ PRINCIPAL =====
1. Agregar país
2. Actualizar datos de un país
3. Buscar país por nombre
4. Filtrar países por continente
5. Filtrar países por rango de población
6. Filtrar países por rango de superficie
7. Ordenar países por nombre
8. Ordenar países por población
9. Ordenar países por superficie
10. Mostrar estadísticas
11.Mostar todos los paises cargados
0. Salir
Seleccione una opción: 1

--- Agregar un nuevo país ---

Ingrese el nombre del país (o 'salir' para cancelar): Dinamarca
Ingrese la población (o 'salir' para cancelar): 12000000
Ingrese la superficie en km² (o 'salir' para cancelar): 100000
Ingrese el continente (no puede estar vacío ni ser un número): Europa
País 'Dinamarca' agregado correctamente.
```

El usuario ingresa los datos de Dinamarca, y el sistema valida los campos y confirma su agregado exitoso al archivo.

- Validación de entrada

```
========================================

===== MENÚ PRINCIPAL =====
1. Agregar país
2. Actualizar datos de un país
3. Buscar país por nombre
4. Filtrar países por continente
5. Filtrar países por rango de población
6. Filtrar países por rango de superficie
7. Ordenar países por nombre
8. Ordenar países por población
9. Ordenar países por superficie
10. Mostrar estadísticas
11.Mostar todos los paises cargados
0. Salir
Seleccione una opción: 1

--- Agregar un nuevo país ---

Ingrese el nombre del país (o 'salir' para cancelar): 12
Error: el nombre no puede estar vacío ni contener números o símbolos.

Ingrese el nombre del país (o 'salir' para cancelar):
Error: el nombre no puede estar vacío ni contener números o símbolos.
```

El sistema valida el campo y muestra un mensaje de error, indicando que el nombre no puede contener números ni estar vacío.

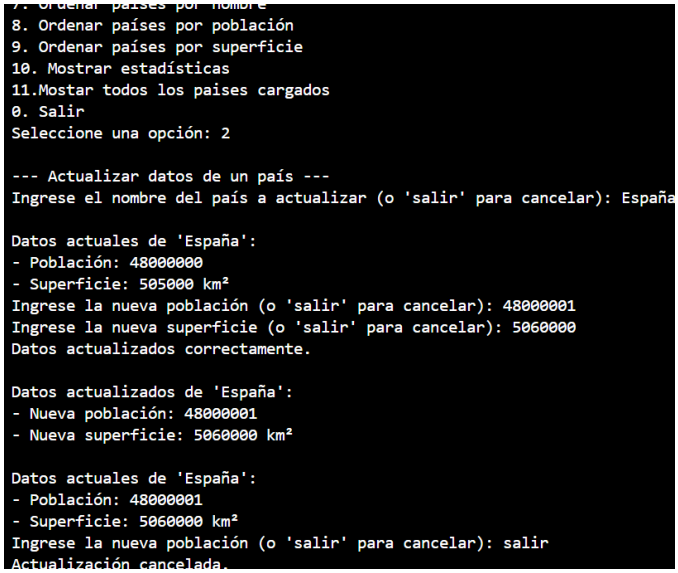
- País ya registrado

```
===== MENÚ PRINCIPAL =====
1. Agregar país
2. Actualizar datos de un país
3. Buscar país por nombre
4. Filtrar países por continente
5. Filtrar países por rango de población
6. Filtrar países por rango de superficie
7. Ordenar países por nombre
8. Ordenar países por población
9. Ordenar países por superficie
10. Mostrar estadísticas
11.Mostar todos los paises cargados
0. Salir
Seleccione una opción: 1

--- Agregar un nuevo país ---

Ingrese el nombre del país (o 'salir' para cancelar): España
Error: ese país ya está registrado.
```

El sistema detecta que el país ingresado (“España”) ya existe en el archivo CSV y muestra un mensaje de error evitando el duplicado.

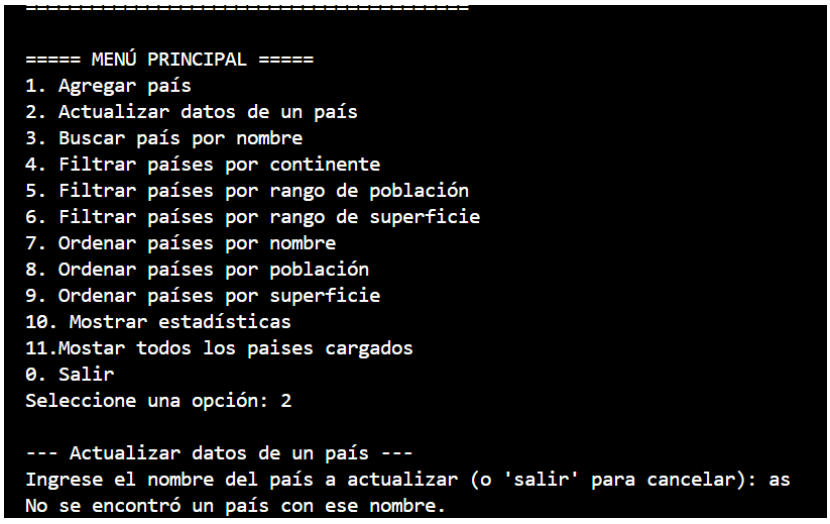
Func 2. Actualizar datos de un país

- Actualización correcta



El sistema muestra los datos actuales, permite editar población y superficie, y confirma la actualización exitosa o cancelada según el ingreso del usuario.

- País no ingresado



El usuario ingresa un nombre inexistente (“as”) y el sistema responde correctamente que no se encontró ningún país con ese nombre.

- Validación de entrada

```
===== MENÚ PRINCIPAL =====
1. Agregar país
2. Actualizar datos de un país
3. Buscar país por nombre
4. Filtrar países por continente
5. Filtrar países por rango de población
6. Filtrar países por rango de superficie
7. Ordenar países por nombre
8. Ordenar países por población
9. Ordenar países por superficie
10. Mostrar estadísticas
11.Mostar todos los paises cargados
0. Salir
Seleccione una opción: 2

--- Actualizar datos de un país ---
Ingrese el nombre del país a actualizar (o 'salir' para cancelar): España

Datos actuales de 'España':
- Población: 48000001
- Superficie: 5060000 km²
Ingrese la nueva población (o 'salir' para cancelar): aa
Error: población inválida. Debe ser un número entre 1 y 2.000.000.000.
Ingrese la nueva población (o 'salir' para cancelar):
```

El usuario ingresa un valor no numérico (“aa”) y el sistema valida el error, mostrando un mensaje que exige un número dentro del rango permitido.

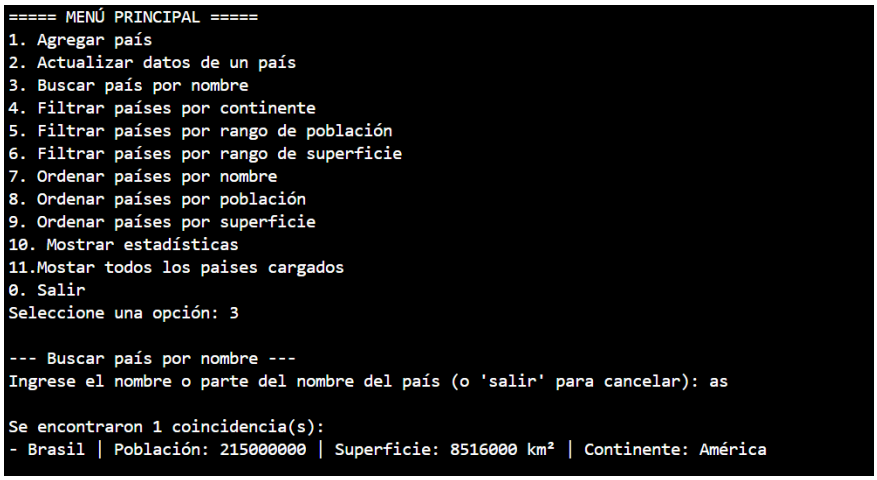
Func 3. Buscar pais por nombre

- Pais encontrado

```
===== MENÚ PRINCIPAL =====
1. Agregar país
2. Actualizar datos de un país
3. Buscar país por nombre
4. Filtrar países por continente
5. Filtrar países por rango de población
6. Filtrar países por rango de superficie
7. Ordenar países por nombre
8. Ordenar países por población
9. Ordenar países por superficie
10. Mostrar estadísticas
11.Mostar todos los paises cargados
0. Salir
Seleccione una opción: 3

--- Buscar país por nombre ---
Ingrese el nombre o parte del nombre del país (o 'salir' para cancelar): España

Se encontraron 1 coincidencia(s):
- España | Población: 48000001 | Superficie: 5060000 km² | Continente: Europa
```

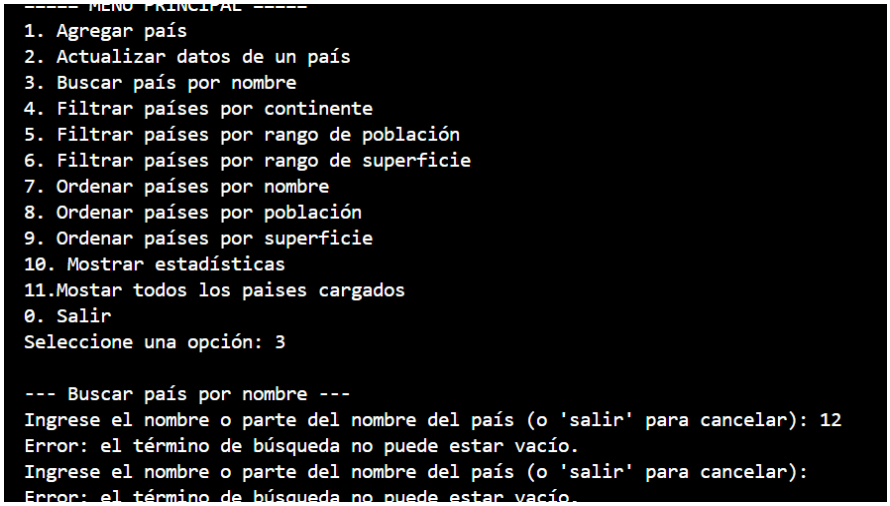
El usuario ingresa “España” y el sistema muestra la coincidencia exacta con todos sus datos registrados.

- País no encontrado o con bajas similitudes



El sistema realiza una búsqueda parcial y muestra correctamente el resultado coincidente: Brasil.

- Validación de entrada



El sistema valida correctamente la entrada y muestra mensajes de error, indicando que el término no puede ser vacío ni numérico.

Func 4. Filtrar países por continente

- Continente existente

```
0. Salir
Seleccione una opción: 4

--- Filtrar países por continente ---
Ingrese el nombre del continente (o 'salir' para cancelar): America

Se encontraron 14 país(es) en America:
- Argentina | Población: 47000000 | Superficie: 2780000 km²
- Brasil | Población: 215000000 | Superficie: 8516000 km²
- Chile | Población: 19000000 | Superficie: 756000 km²
- Uruguay | Población: 3500000 | Superficie: 176000 km²
- Paraguay | Población: 7200000 | Superficie: 406000 km²
- Bolivia | Población: 121000000 | Superficie: 110000 km²
- Perú | Población: 34000000 | Superficie: 1285000 km²
- Colombia | Población: 52000000 | Superficie: 1142000 km²
- Venezuela | Población: 28000000 | Superficie: 916000 km²
- Ecuador | Población: 18000000 | Superficie: 283000 km²
- México | Población: 126000000 | Superficie: 1973000 km²
- Estados Unidos | Población: 331000000 | Superficie: 9834000 km²
- Canadá | Población: 39000000 | Superficie: 9985000 km²
- Cuba | Población: 1000 | Superficie: 2000 km²
```

El sistema muestra correctamente los 14 países pertenecientes a ese continente con sus datos de población y superficie.

- Continente inexistente o sin países registrados

```
===== MENÚ PRINCIPAL =====
1. Agregar país
2. Actualizar datos de un país
3. Buscar país por nombre
4. Filtrar países por continente
5. Filtrar países por rango de población
6. Filtrar países por rango de superficie
7. Ordenar países por nombre
8. Ordenar países por población
9. Ordenar países por superficie
10. Mostrar estadísticas
11.Mostar todos los paises cargados
0. Salir
Seleccione una opción: 4

--- Filtrar países por continente ---
Ingrese el nombre del continente (o 'salir' para cancelar): a
No se encontraron países en ese continente.
```

El sistema valida el texto y muestra que no se encontraron países correspondientes a ese continente.

- Validación de entrada

```
===== MENÚ PRINCIPAL =====
1. Agregar país
2. Actualizar datos de un país
3. Buscar país por nombre
4. Filtrar países por continente
5. Filtrar países por rango de población
6. Filtrar países por rango de superficie
7. Ordenar países por nombre
8. Ordenar países por población
9. Ordenar países por superficie
10. Mostrar estadísticas
11.Mostar todos los paises cargados
0. Salir
Seleccione una opción: 4

--- Filtrar países por continente ---
Ingrese el nombre del continente (o 'salir' para cancelar): 2
Error: el continente no puede ser un número.
```

El usuario ingresa un número (“2”) y el sistema valida la entrada, mostrando un mensaje de error que indica que el continente no puede ser numérico.

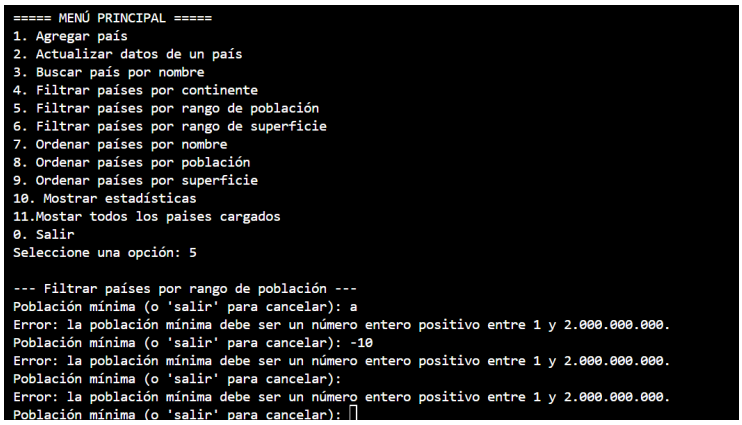
Func 5. Filtrar países por rango de población

- Filtrado correcto

```
Seleccione una opción: 5

--- Filtrar países por rango de población ---
Población mínima (o 'salir' para cancelar): 100000
Población máxima (o 'salir' para cancelar): 100000000

Se encontraron 19 país(es) con población entre 100000 y 100000000:
- Argentina | Población: 47000000 | Superficie: 2780000 km² | Continente: América
- Chile | Población: 19000000 | Superficie: 756000 km² | Continente: América
- Uruguay | Población: 3500000 | Superficie: 176000 km² | Continente: América
- Paraguay | Población: 7200000 | Superficie: 406000 km² | Continente: América
- Perú | Población: 34000000 | Superficie: 1285000 km² | Continente: América
- Colombia | Población: 52000000 | Superficie: 1142000 km² | Continente: América
- Venezuela | Población: 28000000 | Superficie: 916000 km² | Continente: América
- Ecuador | Población: 18000000 | Superficie: 283000 km² | Continente: América
- Canadá | Población: 39000000 | Superficie: 9985000 km² | Continente: América
- España | Población: 48000001 | Superficie: 5060000 km² | Continente: Europa
- Francia | Población: 67000000 | Superficie: 551000 km² | Continente: Europa
- Alemania | Población: 83000000 | Superficie: 357000 km² | Continente: Europa
- Italia | Población: 59000000 | Superficie: 301000 km² | Continente: Europa
- Reino Unido | Población: 67000000 | Superficie: 243000 km² | Continente: Europa
- Portugal | Población: 10300000 | Superficie: 92200 km² | Continente: Europa
- Noruega | Población: 5400000 | Superficie: 385000 km² | Continente: Europa
- Finlandia | Población: 5500000 | Superficie: 338000 km² | Continente: Europa
- Australia | Población: 26000000 | Superficie: 7692000 km² | Continente: Oceanía
- Dinamarca | Población: 12000000 | Superficie: 100000 km² | Continente: Europa
```

El sistema muestra los 19 países cuya población está entre 10.000 y 100.000.000, detallando sus datos completos.

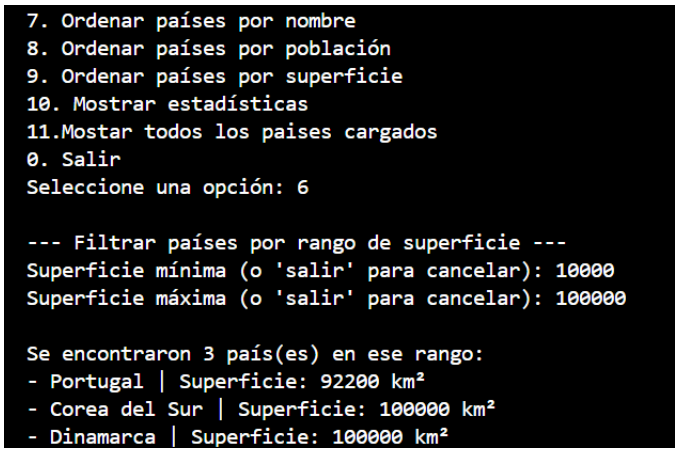
- Validación de entrada



El sistema valida los errores y muestra mensajes aclarando que la población debe ser un número entero positivo dentro del rango permitido.

Func 6. Filtrar países por rango de superficie

- Filtrado correcto



El sistema filtra correctamente y muestra los tres países que cumplen con ese rango: Portugal, Corea del Sur y Dinamarca.

- Validación de entrada

```
3. Buscar país por nombre
4. Filtrar países por continente
5. Filtrar países por rango de población
6. Filtrar países por rango de superficie
7. Ordenar países por nombre
8. Ordenar países por población
9. Ordenar países por superficie
10. Mostrar estadísticas
11.Mostar todos los paises cargados
0. Salir
Seleccione una opción: 6

--- Filtrar países por rango de superficie ---
Superficie mínima (o 'salir' para cancelar): a
Error: la superficie mínima debe ser un número entero positivo entre 1 y 20.000.000.
Superficie mínima (o 'salir' para cancelar): -1
Error: la superficie mínima debe ser un número entero positivo entre 1 y 20.000.000.
Superficie mínima (o 'salir' para cancelar):
Error: la superficie mínima debe ser un número entero positivo entre 1 y 20.000.000.
Superficie mínima (o 'salir' para cancelar):
```

El programa valida los errores de entrada, mostrando mensajes que exigen un número entero positivo dentro del rango permitido.

Func 7. Ordenar países por nombre

Se ejecuta la opción “Ordenar países por nombre”, mostrando el listado en orden alfabético ascendente.

El sistema imprime correctamente todos los países con sus datos de población, superficie y continente.

```
--- Ordenar países por nombre ---

Listado ordenado por nombre (ascendente):
- Alemania | Población: 83000000 | Superficie: 357000 km² | Continente: Europa
- Argentina | Población: 47000000 | Superficie: 2780000 km² | Continente: América
- Australia | Población: 26000000 | Superficie: 7692000 km² | Continente: Oceanía
- Bolivia | Población: 121000000 | Superficie: 110000 km² | Continente: América
- Brasil | Población: 215000000 | Superficie: 8516000 km² | Continente: América
- Canadá | Población: 39000000 | Superficie: 9985000 km² | Continente: América
- Chile | Población: 19000000 | Superficie: 756000 km² | Continente: América
- China | Población: 1410000000 | Superficie: 9600000 km² | Continente: Asia
- Colombia | Población: 52000000 | Superficie: 1142000 km² | Continente: América
- Corea del Norte | Población: 1000 | Superficie: 1000 km² | Continente: Asia
- Corea del Sur | Población: 521000000 | Superficie: 100000 km² | Continente: Asia
- Cuba | Población: 1000 | Superficie: 2000 km² | Continente: America
- Dinamarca | Población: 12000000 | Superficie: 100000 km² | Continente: Europa
- Ecuador | Población: 18000000 | Superficie: 283000 km² | Continente: América
- España | Población: 48000001 | Superficie: 5060000 km² | Continente: Europa
- Estados Unidos | Población: 331000000 | Superficie: 9834000 km² | Continente: América
- Finlandia | Población: 5500000 | Superficie: 338000 km² | Continente: Europa
- Francia | Población: 67000000 | Superficie: 551000 km² | Continente: Europa
- India | Población: 1400000000 | Superficie: 3287000 km² | Continente: Asia
```

Func 8. Ordenar países por población

Se elige la opción “Ordenar países por población”, organizando el listado de forma descendente.

El sistema muestra correctamente los países del más poblado al menos poblado, con todos sus datos asociados.

```
Seleccione una opcion: 8

--- Ordenar países por poblacion ---

Listado ordenado por poblacion (descendente):
- China | Población: 1410000000 | Superficie: 9600000 km² | Continente: Asia
- India | Población: 1400000000 | Superficie: 3287000 km² | Continente: Asia
- Corea del Sur | Población: 521000000 | Superficie: 100000 km² | Continente: Asia
- Estados Unidos | Población: 331000000 | Superficie: 9834000 km² | Continente: América
- Brasil | Población: 215000000 | Superficie: 8516000 km² | Continente: América
- Rusia | Población: 143000000 | Superficie: 17098000 km² | Continente: Europa
- México | Población: 126000000 | Superficie: 1973000 km² | Continente: América
- Japón | Población: 125000000 | Superficie: 377000 km² | Continente: Asia
- Bolivia | Población: 121000000 | Superficie: 110000 km² | Continente: América
- Alemania | Población: 83000000 | Superficie: 357000 km² | Continente: Europa
- Francia | Población: 67000000 | Superficie: 551000 km² | Continente: Europa
- Reino Unido | Población: 67000000 | Superficie: 243000 km² | Continente: Europa
- Italia | Población: 59000000 | Superficie: 301000 km² | Continente: Europa
- Colombia | Población: 52000000 | Superficie: 1142000 km² | Continente: América
- España | Población: 48000001 | Superficie: 5060000 km² | Continente: Europa
- Argentina | Población: 47000000 | Superficie: 2780000 km² | Continente: América
```

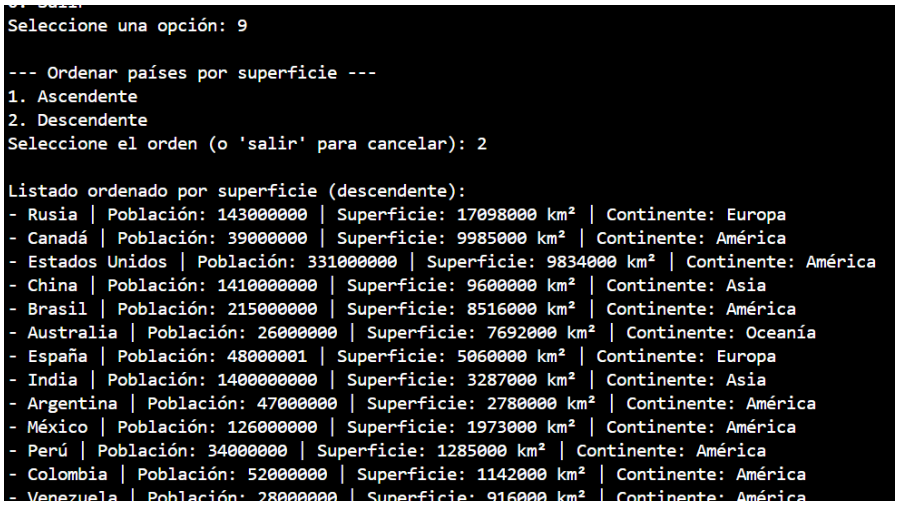
Func 9. Ordenar países por superficie

- Ascendente

```
Seleccione una opción: 9

--- Ordenar países por superficie ---
1. Ascendente
2. Descendente
Seleccione el orden (o 'salir' para cancelar): 1

Listado ordenado por superficie (ascendente):
- qatar | Población: 1000 | Superficie: 1000 km² | Continente: Asia
- Corea del Norte | Población: 1000 | Superficie: 1000 km² | Continente: Asia
- Cuba | Población: 1000 | Superficie: 2000 km² | Continente: America
- Portugal | Población: 10300000 | Superficie: 92200 km² | Continente: Europa
- Corea del Sur | Población: 521000000 | Superficie: 100000 km² | Continente: Asia
- Dinamarca | Población: 12000000 | Superficie: 100000 km² | Continente: Europa
- Bolivia | Población: 121000000 | Superficie: 110000 km² | Continente: América
- Uruguay | Población: 3500000 | Superficie: 176000 km² | Continente: América
- Reino Unido | Población: 67000000 | Superficie: 243000 km² | Continente: Europa
- Ecuador | Población: 18000000 | Superficie: 283000 km² | Continente: América
- Italia | Población: 59000000 | Superficie: 301000 km² | Continente: Europa
- Finlandia | Población: 5500000 | Superficie: 338000 km² | Continente: Europa
- Alemania | Población: 83000000 | Superficie: 357000 km² | Continente: Europa
- Japón | Población: 125000000 | Superficie: 377000 km² | Continente: Asia
- Noruega | Población: 5400000 | Superficie: 385000 km² | Continente: Europa
- Paraguay | Población: 7200000 | Superficie: 406000 km² | Continente: América
```

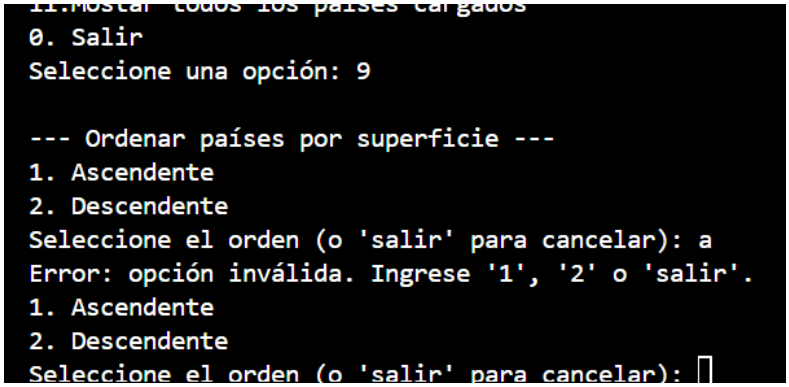
El sistema organiza correctamente los países de menor a mayor superficie, mostrando sus datos completos.

- Descendente



El sistema muestra el listado correctamente del país más grande al más pequeño, junto con sus datos de población y continente.}

- Validación de entrada



El sistema valida la opción incorrecta y muestra un mensaje de error, solicitando una entrada válida (“1”, “2” o “salir”).

Func 10. Mostrar estadísticas

```
0. Salir
Seleccione una opción: 10

--- Estadísticas de países ---
País con mayor población: China (1,410,000,000)
País con menor población: qatar (1,000)
Promedio de población: 162,029,129
Promedio de superficie: 2,701,522 km²
Cantidad de países por continente:
- America: 14
- Asia: 6
- Europa: 10
- Oceania: 1
```

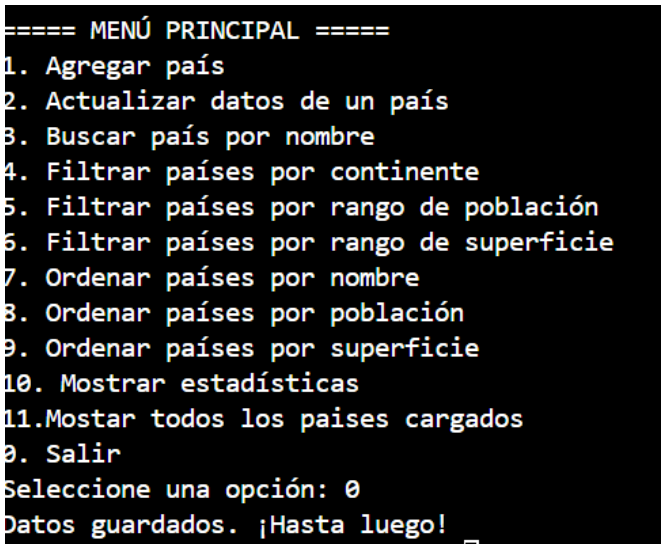
El sistema calcula y muestra el país con mayor y menor población, los promedios generales y la cantidad de países por continente.

Func 11. Listado de países

```
Seleccione una opción: 11

--- Listado completo de países ---
- Argentina | Población: 47000000 | Superficie: 2780000 km² | Continente: América
- Brasil | Población: 215000000 | Superficie: 8516000 km² | Continente: América
- Chile | Población: 19000000 | Superficie: 756000 km² | Continente: América
- Uruguay | Población: 3500000 | Superficie: 176000 km² | Continente: América
- Paraguay | Población: 7200000 | Superficie: 406000 km² | Continente: América
- Bolivia | Población: 121000000 | Superficie: 110000 km² | Continente: América
- Perú | Población: 34000000 | Superficie: 1285000 km² | Continente: América
- Colombia | Población: 52000000 | Superficie: 1142000 km² | Continente: América
- Venezuela | Población: 28000000 | Superficie: 916000 km² | Continente: América
- Ecuador | Población: 18000000 | Superficie: 283000 km² | Continente: América
- México | Población: 126000000 | Superficie: 1973000 km² | Continente: América
- Estados Unidos | Población: 331000000 | Superficie: 9834000 km² | Continente: América
- Canadá | Población: 39000000 | Superficie: 9985000 km² | Continente: América
- España | Población: 48000001 | Superficie: 5060000 km² | Continente: Europa
- Francia | Población: 67000000 | Superficie: 551000 km² | Continente: Europa
```

El sistema imprime el listado completo de países registrados, con su población, superficie y continente correspondiente.

Func 0. Salir



Se finaliza el programa.

El sistema guarda todos los datos en el archivo CSV y muestra el mensaje de despedida “¡Hasta luego!”.

Conclusiones Grupales de aprendizaje:

1. Aprendimos a usar listas y diccionarios en serio. Al principio parecía complicado, pero al trabajar con el dataset de países vimos que son súper útiles para organizar y manejar datos.
2. Nos ordenamos mejor con funciones. Dividir el código en funciones nos ayudó a no enredarnos y a que cada parte del programa tenga su tarea clara. Eso hizo que todo sea más fácil de leer y mantener.
3. Nos dimos cuenta de la importancia de validar datos, cuando probamos el sistema, vimos que sin controles se rompía fácil. Aprendimos a poner validaciones y mensajes claros para que el programa sea más confiable.
4. Filtrar, ordenar y sacar estadísticas fue un buen ejercicio nos sirvió para entender cómo aplicar lógica y algoritmos básicos en algo práctico. Además, ver los resultados (como el país con más población) nos motivó.
5. El trabajo en equipo fue clave ,nos repartimos tareas, usamos GitHub para organizarnos y nos apoyamos mutuamente cuando algo no salía. Eso hizo que el proyecto avance mejor y que todos aprendamos.
6. El TPI nos dejó experiencia real.No fue solo teoría: armamos un programa completo, con informe, código y video. Eso nos dio confianza y nos mostró cómo sería un proyecto en la vida laboral.

7. Fuimos encontrando nuestro propio orden de trabajo .Al principio costaba bastante organizarnos y decidir por dónde empezar, pero con la práctica fuimos mejorando y ahora sentimos que avanzamos de manera más clara y ordenada.
8. Nos manejamos mejor con controles cruzados, cada uno revisa la parte del otro y avisa si encuentra algún error. Esto nos ayudó a detectar fallos más rápido y a mejorar la calidad del trabajo en equipo.

Fuentes:

- https://docs.python.org/3/library/csv.html
- https://realpython.com/python-csv/
- Python Software Foundation. (2025). Tutorial de Python: Estructuras de datos — Listas. Python 3.14.0 documentation. Recuperado de https://docs.python.org/es/3/tutorial/datastructures.html
- Python Software Foundation. (2025). Tutorial de Python: Control de flujo — Condicionales. Python 3.14.0 documentation. Recuperado de https://docs.python.org/es/3/tutorial/controlflow.html
- Python Software Foundation. (2025). Objetos diccionario. Python 3.14.0 documentation. Recuperado de https://docs.python.org/es/3/c-api/dict.html
- Python Software Foundation. (2025). Funciones en Python. Python 3.14.0 documentation. Recuperado de https://docs.python.org/es/3/library/functions.html
- https://docs.python.org/3/howto/sorting.html
- https://docs.python.org/3/library/stdtypes.html#list.sort
- https://docs.python.org/3/library/functions.html#sorted
- https://docs.python.org/3/library/statistics.html
- https://realpython.com/python-statistics/