Trabajo Práctico Integrador

Alumnos - Comisión Ag25-1C-03

Castilla Cesia | Castillo Alfredo

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Programación 1

Docente Titular

Cinthia Rigoni

Docente Tutor

Brian Lara

9 de Noviembre de 2025

Tabla de contenido

Introducción

Este documento presenta el Trabajo Práctico Integrador de la materia Programación 1, correspondiente a la Tecnicatura Universitaria en Programación a Distancia (TUPAD) de la Universidad Tecnológica Nacional (UTN).

El proyecto consiste en el desarrollo de un Sistema de Gestión de Datos de Países implementado en Python, que permite realizar operaciones de consulta, filtrado, ordenamiento y análisis estadístico sobre un catálogo de países almacenado en formato CSV.

ALCANCE DEL PROYECTO

El sistema desarrollado integra los conceptos fundamentales vistos durante la cursada:

- Estructuras de datos (listas y diccionarios)
- Funciones y modularización
- Estructuras condicionales y repetitivas
- Algoritmos de búsqueda y ordenamiento
- Cálculo de estadísticas básicas
- Persistencia de datos en archivos CSV
- Validación de datos sin uso de excepciones

El proyecto cumple con todas las restricciones académicas establecidas: sin uso de funciones lambda, sin manejo de excepciones (try-except), sin clases, y sin recursividad.

Objetivo del trabajo

Desarrollar una aplicación en Python que permita gestionar información sobre países del mundo, aplicando los conceptos fundamentales de programación vistos durante la cursada de Programación 1, con énfasis en estructuras de datos, modularización, algoritmos de búsqueda y ordenamiento, y persistencia de datos.

OBJETIVOS ESPECÍFICOS

1. Implementar un sistema CRUD (Create, Read, Update, Delete) para la gestión de datos de países.
2. Aplicar estructuras de datos adecuadas (listas y diccionarios) para el almacenamiento y manipulación de información.
3. Desarrollar una arquitectura modular que separe responsabilidades en módulos especializados.
4. Implementar algoritmos de búsqueda (exacta y parcial) para localizar países específicos.
5. Crear sistemas de filtrado por múltiples criterios (continente, población, superficie).
6. Implementar algoritmos de ordenamiento por diferentes atributos (nombre, población, superficie).
7. Calcular estadísticas descriptivas sobre los datos (máximos, mínimos, promedios, conteos).
8. Gestionar la persistencia de datos mediante archivos CSV con lectura y escritura.
9. Validar datos de entrada de forma robusta sin utilizar manejo de excepciones.
10. Desarrollar una interfaz de usuario interactiva mediante menú en consola.

ALCANCE Y RESTRICCIONES

Alcance:

- Sistema funcional con 8 opciones de menú principales

• 8 módulos especializados (datos, búsquedas, filtros, ordenamiento, estadísticas, utilidades, validaciones)

• Catálogo de países con 4 atributos: nombre, población, superficie y continente

• Persistencia automática de cambios en archivo CSV

Restricciones Académicas:

• Sin uso de funciones lambda (se utilizan funciones con nombre explícito)

• Sin manejo de excepciones try-except (validación manual con condicionales)

• Sin uso de clases (programación funcional)

• Sin recursividad (todos los algoritmos son iterativos)

Marco teórico

1. INTRODUCCIÓN

En la era de la información, la gestión eficiente de datos es una competencia fundamental para cualquier profesional en tecnología. Este trabajo práctico aborda el desarrollo de una aplicación que integra conceptos clave de Programación 1, permitiendo consolidar la comprensión de estructuras de datos, algoritmos básicos y buenas prácticas de programación.

1.1. Objetivo del Proyecto

El objetivo principal es desarrollar una aplicación en Python que permita gestionar información sobre países del mundo, aplicando:

- Lectura y escritura de archivos CSV
- Estructuras de datos (listas y diccionarios)
- Funciones modulares
- Algoritmos de búsqueda, filtrado y ordenamiento
- Validación de datos
- Cálculo de estadísticas

1.2. Contexto del Dominio

Los países constituyen un dominio de datos complejo pero familiar, permitiendo enfocar la atención en las técnicas de programación sin necesidad de entender un dominio completamente desconocido.

Cada país en el sistema es representado por cuatro atributos:

- Nombre (string): Identificador único del país
- Población (int): Número de habitantes
- Superficie (int): Extensión territorial en km²
- Continente (string): Ubicación geográfica

2. ESTRUCTURAS DE DATOS

2.1. Listas (Lists)

Una lista es una colección ordenada de elementos que pueden ser de diferentes tipos. En Python se declaran con corchetes [].

Características principales:

- Ordenadas: Mantienen el orden de inserción de los elementos
- Mutables: Pueden ser modificadas después de su creación
- Indexadas: Se accede a elementos por posición (índices comienzan en 0)
- Heterogéneas: Pueden contener diferentes tipos de datos

Ejemplo en el proyecto:

```
# Lista de diccionarios que representa el catálogo de países
    paises = [
        {'nombre': 'Argentina', 'poblacion': 45376763, 'superficie': 2780400, 'continente': 'America'},
        {'nombre': 'Brasil', 'poblacion': 213993437, 'superficie': 8515767, 'continente': 'America'},
        {'nombre': 'Japon', 'poblacion': 125800000, 'superficie': 377975, 'continente': 'Asia'}
    ]
```

Operaciones fundamentales utilizadas en el proyecto:

```
# Agregar elemento al final
    paises.append(nuevo_pais)

# Acceder por índice
    primer_pais = paises[0]

# Iteración sobre elementos
    for pais in paises:
      print(pais['nombre'])

# Obtener longitud
    cantidad_paises = len(paises)

# Filtrado con list comprehension
    paises_america = [p for p in paises if p['continente'] == 'America']
```

Ventajas de usar listas:

- Acceso rápido por índice: O(1)
- Facilidad de uso y sintaxis intuitiva
- Flexible para almacenar colecciones ordenadas

Desventajas:

- Búsqueda lineal es O(n)

- No apropiada para búsquedas frecuentes sin ordenamiento

Aplicación en el proyecto: Las listas son la estructura central para almacenar todos los países en memoria durante la ejecución del programa.

2.2. Diccionarios (Dictionaries)

Un diccionario es una colección de pares clave-valor. Se declara con llaves {}.

Características principales:

- Mapeo de claves a valores
- Mutables: Se pueden agregar, modificar o eliminar pares
- Acceso por clave: No requiere conocer la posición
- Claves únicas: No puede haber claves duplicadas

Ejemplo en el proyecto:

```
# Diccionario que representa un país individual
    pais = {
       'nombre': 'Argentina',
       'poblacion': 45376763,
       'superficie': 2780400,
       'continente': 'America'
    }

# Acceso a valores
    nombre = pais['nombre']              # 'Argentina'
    poblacion = pais.get('poblacion')    # 45376763
```

Operaciones fundamentales:

```
# Agregar o actualizar valor
    pais['capital'] = 'Buenos Aires'

# Acceso seguro (retorna None si no existe)
    gdp = pais.get('gdp', 0)

# Iteración por claves y valores
    for clave, valor in pais.items():
        print(f"{clave}: {valor}")

# Verificar existencia de clave
    if 'nombre' in pais:
        print("El país tiene nombre")
```

Ventajas de usar diccionarios:

- Acceso O(1) por clave (muy rápido)
- Código más legible con claves descriptivas
- Ideal para representar entidades con propiedades

Desventajas:

- Mayor consumo de memoria que listas simples
- No mantienen orden en versiones antiguas de Python (< 3.7)

Aplicación en el proyecto: Cada país individual es representado como un diccionario, permitiendo acceso intuitivo a sus propiedades mediante claves descriptivas.

2.3. Combinación: Lista de Diccionarios

La estructura fundamental del proyecto combina listas y diccionarios:

```
paises = [
    {'nombre': 'Argentina', 'poblacion': 45376763, 'superficie': 2780400, 'continente': 'America'},
    {'nombre': 'Brasil', 'poblacion': 213993437, 'superficie': 8515767, 'continente': 'America'},
    {'nombre': 'Japon', 'poblacion': 125800000, 'superficie': 377975, 'continente': 'Asia'}
]
```

Ventajas de esta combinación:

- Acceso por índice: paises[0] obtiene el primer país
- Acceso por clave: paises[0]['nombre'] obtiene el nombre del primer país
- Iteración natural sobre la colección
- Representa perfectamente datos tabulares (similar a una tabla de base de datos)

3. FUNCIONES Y MODULARIZACIÓN

3.1. Concepto de Función

Una función es un bloque de código reutilizable que realiza una tarea específica. Las funciones promueven:

- Reutilización: Código escrito una vez, usado múltiples veces
- Mantenibilidad: Cambios centralizados en un solo lugar
- Legibilidad: Código organizado y comprensible

• Prueba: Fácil de testear funcionalidades de forma independiente

Estructura básica de una función en Python:

```
def nombre_funcion(parametros):
"""Documentación de la función (docstring)"""
# Cuerpo de la función
    resultado = realizar_operacion(parametros)
    return resultado
```

3.2. Modularización en el Proyecto

El proyecto sigue el principio fundamental: "Una función = Una responsabilidad"

La aplicación está organizada en 8 módulos especializados:

tp_integrador.py

(Archivo Principal)

módulos	
datos.py	Persistencia CSV
busquedas.py	Búsqueda de países
estadisticas.py	Cálculos estadísticos
filtros.py	Filtrado de datos
ordenamiento.py	Ordenamiento
utilidades.py	Presentación
validaciones.py	Validaciones

3.2.1. Módulo de Datos (datos.py)

Responsabilidad: Gestión de persistencia y operaciones CRUD

Funciones principales:

- cargar_desde_csv(): Lee el archivo CSV y carga los datos en memoria
- guardar_en_csv(): Persiste los cambios en el archivo CSV
- agregar_pais(): Agrega un nuevo país al catálogo
- actualizar_pais(): Modifica población y superficie de un país existente
- obtener_todos(): Retorna la lista completa de países
- buscar_por_nombre(): Localiza un país específico

Ejemplo de función:

```
def agregar_pais(paises, nombre, poblacion, superficie, continente):
    """
    Agrega un nuevo país al catálogo.
    Args:
    paises: Lista de diccionarios con los países
    nombre: Nombre del país
    poblacion: Número de habitantes
    superficie: Extensión en km²
    continente: Continente del país
    Returns:
    El diccionario del país agregado
    """
      nuevo_pais = {
```

```
        'nombre': nombre,
        'poblacion': poblacion,
        'superficie': superficie,
        'continente': continente
    }
    paises.append(nuevo_pais)
    return nuevo_pais
```

3.2.2. Módulo de Búsquedas (busquedas.py)

Responsabilidad: Localizar países según diferentes criterios

Funciones principales:

- buscar_exacto(): Búsqueda exacta por nombre (case-insensitive)
- buscar_parcial(): Búsqueda por coincidencia parcial

Ejemplo:

```
def buscar_exacto(paises, nombre):
    """
    Busca un país por nombre exacto (insensible a mayúsculas).

    Args:
        paises: Lista de países
        nombre: Nombre a buscar

    Returns:
        Diccionario del país si se encuentra, None si no existe
```

```
    """
    nombre_lower = nombre.lower()
    for pais in paises:
        if pais['nombre'].lower() == nombre_lower:
            return pais
    return None
```

3.2.3. Módulo de Filtros (filtros.py)

Responsabilidad: Obtener subconjuntos de datos según criterios

Funciones principales:

- filtrar_por_continente(): Filtra países de un continente específico
- filtrar_por_rango_poblacion(): Filtra por rango de población
- filtrar_por_rango_superficie(): Filtra por rango de superficie

Ejemplo:

```
def filtrar_por_continente(paises, continente):
    """
    Filtra países pertenecientes a un continente específico.
    Args:
        paises: Lista de países
        continente: Nombre del continente

    Returns:
        Lista de países del continente especificado
    """
```

```
    continente_lower = continente.lower()
    return [p for p in paises if p['continente'].lower() == continente_lower]
```

3.2.4. Módulo de Ordenamiento (ordenamiento.py)

Responsabilidad: Reorganizar datos según diferentes criterios

Funciones principales:

- ordenar_por_nombre(): Ordena alfabéticamente
- ordenar_por_poblacion(): Ordena por número de habitantes
- ordenar_por_superficie(): Ordena por extensión territorial

Ejemplo:

```
def ordenar_por_poblacion(paises, descendente=False):
    """
    Ordena países por población.

    Args:
        paises: Lista de países
        descendente: Si True, ordena de mayor a menor

    Returns:
        Lista ordenada de países
    """
    def obtener_poblacion(pais):
        return pais['poblacion']

    return sorted(paises, key=obtener_poblacion, reverse=descendente)
```

3.2.5. Módulo de Estadísticas (estadisticas.py)

Responsabilidad: Calcular indicadores y métricas

Funciones principales:

- pais_mayor_poblacion(): Encuentra el país más poblado
- pais_menor_poblacion(): Encuentra el país menos poblado
- promedio_poblacion(): Calcula la población promedio
- promedio_superficie(): Calcula la superficie promedio
- contar_por_continente(): Cuenta países por continente

3.2.6. Módulo de Utilidades (utilidades.py)

Responsabilidad: Presentación y formateo de datos

Funciones principales:

- mostrar_tabla_paises(): Presenta datos en formato tabular
- mostrar_pais_individual(): Muestra información detallada de un país
- formatear_numero(): Formatea números con separadores de miles
- limpiar_pantalla(): Limpia la pantalla de la consola

3.2.7. Módulo de Validaciones (validaciones.py)

Responsabilidad: Validar entrada de datos

Funciones principales:

- validar_numero_positivo(): Verifica que un valor sea numérico positivo
- validar_texto_no_vacio(): Verifica que un texto no esté vacío
- validar_rango(): Verifica que un valor esté dentro de un rango

3.3. Beneficios de la Modularización

Beneficio	Explicación
Reutilización contextos	Las funciones pueden usarse en múltiples sin duplicar código
Mantenimiento	Cambios localizados no afectan todo el sistema Más fácil encontrar y corregir errores
Legibilidad	Código autoexplicativo y fácil de comprender Nombres de funciones describen su propósito
Escalabilidad	Agregar nuevas funcionalidades es sencillo sin reestructurar todo el código
Trabajo en equipo	Diferentes personas pueden trabajar en módulos diferentes simultáneamente

4. CONDICIONALES Y CONTROL DE FLUJO

4.1. Estructura if-elif-else

Los condicionales permiten ejecutar diferentes bloques de código según se cumplan o no ciertas condiciones.

Sintaxis básica:

```
if condicion1:
    # Se ejecuta si condicion1 es True
    pass
elif condicion2:
    # Se ejecuta si condicion1 es False y condicion2 es True
    pass
else:
    # Se ejecuta si todas las condiciones anteriores son False
```

```
        pass
```

Ejemplo en el proyecto:

```
# Validación de población
    if poblacion <= 0:
        print("Error: La población debe ser positiva")
        return False
    else:
        print("Población válida")
        return True

# Procesamiento de búsqueda
    if resultados:
        mostrar_tabla_paises(resultados)
    else:
        print("No se encontraron países con ese criterio")
```

4.2. Operadores Lógicos

Los operadores lógicos permiten combinar múltiples condiciones:

```
# AND: Ambas condiciones deben ser True
    if poblacion > 0 and superficie > 0:
        agregar_pais(...)

# OR: Al menos una condición debe ser True
```

```
    if continente == 'America' or continente == 'Europa':
        procesar_pais(...)

# NOT: Invierte el valor lógico
    if not nombre.strip():
        print("Error: Nombre vacío")
```

4.3. Operadores de Comparación

Operador	Significado	Ejemplo
==	Igual a	if edad == 18
!=	Diferente de	if nombre != ""
>	Mayor que	if poblacion > 1000000
<	Menor que	if superficie < 500000
>=	Mayor o igual	if edad >= 18
<=	Menor o igual	if nota <= 10
in	Pertenece a	if 'a' in nombre

4.4. Validaciones en el Proyecto

El proyecto implementa validaciones exhaustivas sin uso de excepciones:

```
def validar_numero_positivo(valor, nombre_campo="valor"):
    """
    Valida que un valor sea numérico y positivo.
    Args:
        valor: Valor a validar
        nombre_campo: Nombre del campo (para mensajes de error)
```

```
        Returns:
            Tupla (bool, valor_convertido, mensaje_error)
        """
# Verificar que sea numérico
        if not str(valor).isdigit():
            return (False, 0, f"Error: {nombre_campo} debe ser numérico")

# Convertir a entero
        numero = int(valor)

# Verificar que sea positivo
        if numero <= 0:
            return (False, 0, f"Error: {nombre_campo} debe ser positivo")

        return (True, numero, "")
```

5. ESTRUCTURAS REPETITIVAS

5.1. Ciclo for

El ciclo for itera sobre elementos de una colección (listas, tuplas, rangos, etc.).

Sintaxis básica:

```
    for elemento in coleccion:
# Procesar elemento
        pass
```

Aplicaciones en el proyecto:

```
# Iteración simple sobre países
    for pais in paises:
        print(pais['nombre'])


# Iteración con índice usando enumerate
    for i, pais in enumerate(paises):
        print(f"{i+1}. {pais['nombre']}")


# Iteración sobre rango de números
    for i in range(len(paises)):
        print(paises[i])


# Iteración al cargar datos del CSV
    for fila in lector_csv:
        pais = crear_pais_desde_fila(fila)
        paises.append(pais)
```

5.2. Ciclo while

El ciclo while ejecuta código mientras una condición sea verdadera.

Sintaxis básica:

```
    while condicion:
# Código a ejecutar
    pass
```

Aplicación en el proyecto - Menú interactivo:

```
ejecutando = True
while ejecutando:
    mostrar_menu()
    opcion = input("Seleccione una opción (1-8): ")

    if opcion == "1":
        agregar_pais_menu()
    elif opcion == "2":
        actualizar_pais_menu()
    # ... más opciones ...
    elif opcion == "8":
        print("Saliendo del programa...")
        ejecutando = False
    else:
        print("Opción inválida")
```

5.3. List Comprehension

Las list comprehensions son una forma concisa y pythónica de crear listas.

Sintaxis:

[expresion for elemento in iterable if condicion]

Ejemplos en el proyecto:

```
# Filtro simple
paises_america = [p for p in paises if p['continente'] == 'America']
```

```
# Transformación
    poblaciones = [p['poblacion'] for p in paises]

# Filtro + Transformación
    nombres_grandes = [p['nombre'] for p in paises if p['poblacion'] > 50000000]
```

Equivalencia con ciclo tradicional:

```
# Forma tradicional (más verbosa)
    resultado = []
    for pais in paises:
        if pais['poblacion'] > 50000000:
            resultado.append(pais)

# List comprehension (más concisa)
    resultado = [p for p in paises if p['poblacion'] > 50000000]
```

Ventajas:

- Más legible que loops equivalentes
- Más rápido en ejecución
- Código más "Pythónico" (idiomático)

6. ORDENAMIENTOS Y BÚSQUEDAS

6.1. Búsqueda Lineal

La búsqueda lineal examina cada elemento de la colección secuencialmente hasta encontrar el objetivo.

Implementación en el proyecto:

```
def buscar_por_nombre_exacto(paises, nombre):
    """
    Busca un país por nombre exacto.
    Complejidad: O(n) - tiempo lineal
    """
    nombre_lower = nombre.lower()
    for pais in paises:
        if pais['nombre'].lower() == nombre_lower:
            return pais
    return None
```

Complejidad temporal: O(n) donde n es el número de países

Ventajas:

- Funciona en listas desordenadas
- Simple de implementar
- No requiere preprocesamiento

Desventajas:

- Ineficiente para grandes volúmenes de datos
- Tiempo de búsqueda aumenta linealmente con la cantidad de datos

Variante - Búsqueda Parcial:

```
def buscar_por_nombre_parcial(paises, texto):
    """
    Busca países cuyo nombre contenga el texto especificado.

   Returns:
       Lista de países que coinciden
    """
    texto_lower = texto.lower()
    return [p for p in paises if texto_lower in p['nombre'].lower()]
```

6.2. Ordenamiento con sorted()

Python proporciona la función sorted() que implementa el algoritmo Timsort (combinación de Merge Sort y Insertion Sort).

Sintaxis:

```
sorted(iterable, key=funcion_clave, reverse=booleano)
```

Parámetros:

- iterable: La colección a ordenar
- key: Función para extraer el valor de comparación
- reverse: Si True, ordena en orden descendente

Complejidad temporal: O(n log n)

6.2.1. Ordenamiento por Nombre

```
def ordenar_por_nombre(paises, descendente=False):
    """
    Ordena países alfabéticamente por nombre.

    Args:
        paises: Lista de países
        descendente: Si True, orden Z→A; si False, orden A→Z

    Returns:
        Lista ordenada de países
    """
    def obtener_nombre(pais):
        return pais['nombre'].lower()

    return sorted(paises, key=obtener_nombre, reverse=descendente)
```

Proceso:

1. Define función extractora: obtener_nombre()
2. sorted() usa esta función para cada país
3. Compara los nombres en minúsculas alfabéticamente
4. reverse=False: A→Z (ascendente)
5. reverse=True: Z→A (descendente)

6.2.2. Ordenamiento por Población

```
    def ordenar_por_poblacion(paises, descendente=False):
        """
        Ordena países por población.

        Args:
            paises: Lista de países
            descendente: Si True, mayor a menor; si False, menor a mayor

        Returns:
            Lista ordenada de países
        """
        def obtener_poblacion(pais):
            return pais['poblacion']

        return sorted(paises, key=obtener_poblacion, reverse=descendente)
```

Nota: El proyecto NO utiliza funciones lambda para cumplir con las restricciones académicas. En su lugar, se definen funciones con nombre explícito.

7. ESTADÍSTICAS BÁSICAS

7.1. Máximo y Mínimo

Encontrar el país con mayor o menor valor en un atributo.

```
    def obtener_pais_mayor_poblacion(paises):
        """
        Encuentra el país con mayor población.
```

```
    Returns:
        Diccionario del país más poblado, o None si la lista está vacía
    """
    if not paises:
        return None

    def obtener_poblacion(pais):
        return pais['poblacion']

    return max(paises, key=obtener_poblacion)
```

Uso:

```
mayor = obtener_pais_mayor_poblacion(paises)
if mayor:
    print(f"País más poblado: {mayor['nombre']}")
    print(f"Población: {mayor['poblacion']:,} habitantes")
```

7.2. Promedio (Media Aritmética)

El promedio se calcula sumando todos los valores y dividiendo por la cantidad de elementos.

Fórmula matemática:

Promedio = (Suma de todos los valores) / (Cantidad de valores)

Implementación:

```
def calcular_promedio_poblacion(paises):
    """
```

```
    Calcula la población promedio de los países.

    Returns:
        Promedio de población, o 0 si la lista está vacía
    """
    if not paises:
        return 0

    total = sum(p['poblacion'] for p in paises)
    return total / len(paises)
```

Desglose del cálculo:

1. sum(p['poblacion'] for p in paises): Suma todas las poblaciones
2. len(paises): Cuenta cuántos países hay
3. División: total / cantidad

Ejemplo numérico:

```
paises = [
    {'nombre': 'A', 'poblacion': 100},
    {'nombre': 'B', 'poblacion': 200},
    {'nombre': 'C', 'poblacion': 300}
]

promedio = (100 + 200 + 300) / 3 = 600 / 3 = 200
```

7.3. Conteo Agrupado

Contar cuántos países pertenecen a cada continente.

```
def contar_paises_por_continente(paises):
    """
    Cuenta la cantidad de países por continente.

    Returns:
        Diccionario con continentes como claves y conteos como valores
    """
    conteos = {}

    for pais in paises:
        continente = pais['continente']

        if continente in conteos:
            conteos[continente] += 1
        else:
            conteos[continente] = 1

    return dict(sorted(conteos.items()))
```

Proceso paso a paso:

1. Inicializar diccionario vacío
2. Iterar sobre cada país

3. Extraer el continente del país

4. Si el continente ya existe en conteos: incrementar en 1

5. Si el continente no existe: crear entrada con valor 1

6. Retornar diccionario ordenado alfabéticamente

Ejemplo de resultado:

```
{'Africa': 54,  'America': 35,
  'Asia': 48,  'Europa': 44,
  'Oceania': 14}
```

8. ARCHIVOS CSV

8.1. Formato CSV

CSV (Comma-Separated Values) es un formato de texto para representar datos tabulares.

Características:

- Valores separados por comas
- Primera fila opcional: encabezados de columnas
- Cada línea: un registro
- Formato simple y universal

Ejemplo de archivo paises.csv:

```
nombre,poblacion,superficie,continente
Argentina,45376763,2780400,America
Brasil,213993437,8515767,America
Japon,125800000,377975,Asia
Alemania,83149300,357022,Europa
```

8.2. Lectura de CSV

Python proporciona el módulo csv para trabajar con archivos CSV.

```
import csv


def cargar_desde_csv(ruta_archivo):
    """
    Lee países desde un archivo CSV.

    Args:
        ruta_archivo: Ruta al archivo CSV

    Returns:
        Lista de diccionarios con los países
    """
    paises = []

    with open(ruta_archivo, 'r', encoding='utf-8') as archivo:
        lector = csv.DictReader(archivo)

        for fila in lector:
            pais = {
                'nombre': fila['nombre'],
                'poblacion': int(fila['poblacion']),
                'superficie': int(fila['superficie']),
                'continente': fila['continente']
```

```
            }
            paises.append(pais)

    return paises
```

Flujo de lectura:

1. Abrir archivo en modo lectura ('r')
2. Especificar codificación UTF-8 para caracteres especiales
3. Crear DictReader que convierte filas a diccionarios
4. Primera fila del CSV se usa como nombres de columnas (claves)
5. Cada fila subsecuente se convierte a un diccionario
6. Los valores son strings por defecto, convertir a int cuando necesario
7. El archivo se cierra automáticamente (with statement)

8.3. Escritura de CSV

```
def guardar_en_csv(paises, ruta_archivo):
    """
    Guarda países en un archivo CSV.

    Args:
        paises: Lista de diccionarios con los países
        ruta_archivo: Ruta donde guardar el archivo
    """
    with open(ruta_archivo, 'w', newline='', encoding='utf-8') as archivo:
        campos = ['nombre', 'poblacion', 'superficie', 'continente']
```

```
        escritor = csv.DictWriter(archivo, fieldnames=campos)


        escritor.writeheader()
        escritor.writerows(paises)
```

Flujo de escritura:

1. Abrir archivo en modo escritura ('w')
2. newline='': Previene líneas vacías extras (importante)
3. Especificar encoding='utf-8'
4. Definir nombres de campos (columnas)
5. Crear DictWriter con los nombres de campos
6. Escribir encabezado (primera fila)
7. Escribir todas las filas de datos
8. El archivo se cierra automáticamente

8.4. Verificación de Archivo

Antes de operar con el archivo, es importante verificar su existencia y permisos:

```
    import os


    def verificar_archivo_csv(ruta_archivo):
        """
        Verifica existencia y permisos del archivo CSV.


        Returns:
            Tupla (existe, es_legible, es_escribible)
```

```
    """
    existe = os.path.exists(ruta_archivo)
    es_legible = os.access(ruta_archivo, os.R_OK) if existe else False
    es_escribible = os.access(ruta_archivo, os.W_OK) if existe else False

    return (existe, es_legible, es_escribible)
```

8.5. Ventajas y Limitaciones del CSV

Ventajas:

- Portabilidad: Se abre en cualquier editor de texto
- Compatibilidad: Excel, Google Sheets, bases de datos
- Simplicidad: Formato legible y fácil de entender
- Ligereza: Archivos pequeños, sin overhead
- Estandarización: Formato bien definido (RFC 4180)

Limitaciones:

- No soporta jerarquías complejas (datos anidados)
- Ambigüedad con separadores en los datos
- No mantiene tipos de datos (todo se lee como string)
- No soporta comentarios nativamente
- No apropiado para grandes volúmenes de datos

9. ARQUITECTURA DEL SISTEMA

9.1. Estructura del Proyecto

TP INTEGRADOR/

```
│
├── tp_integrador.py          # Archivo principal - Menú interactivo
│
├── modulos/                  # Paquete de módulos
│   ├── __init__.py           # Inicialización del paquete
│   ├── datos.py              # CRUD y persistencia CSV
│   ├── busquedas.py          # Búsqueda exacta y parcial
│   ├── estadisticas.py       # Cálculos estadísticos
│   ├── filtros.py            # Filtrado por criterios
│   ├── ordenamiento.py       # Ordenamiento de países
│   ├── utilidades.py         # Formateo y presentación
│   └── validaciones.py       # Validación de datos
│
└── datos/
    └── paises.csv            # Base de datos de países
```

9.2. Flujo de Ejecución

1. Inicio del programa (tp_integrador.py)

↓

2. Cargar datos desde paises.csv (modulos/datos.py)

↓

3. Mostrar menú principal (bucle while)

↓

4. Usuario selecciona opción (1-8)

↓

5. Ejecutar funcionalidad correspondiente

```
│
├──→ Agregar país → Validar → Guardar CSV
├──→ Actualizar → Buscar → Modificar → Guardar CSV
├──→ Buscar → Mostrar resultados
```

├──→ Filtrar → Mostrar resultados

├──→ Ordenar → Mostrar resultados ordenados

└──→ Estadísticas → Calcular → Mostrar

↓

6. Volver al menú (hasta seleccionar opción 8: Salir)

9.3. Principios de Diseño Aplicados

Separación de Responsabilidades (SoC):

Cada módulo tiene una única responsabilidad bien definida, facilitando el mantenimiento y la comprensión del código.

Don't Repeat Yourself (DRY):

El código reutilizable se centraliza en funciones, evitando duplicación.

Modularidad:

El sistema está dividido en módulos independientes que pueden ser modificados, probados y reutilizados sin afectar el resto del sistema.

Validación Exhaustiva:

Todas las entradas de usuario son validadas antes de procesarse, sin depender de excepciones.

Interfaz Clara:

El menú proporciona opciones claras y mensajes descriptivos para facilitar el uso del sistema.

10. RESTRICCIONES Y CONSIDERACIONES ACADÉMICAS

10.1. Restricciones Implementadas

El proyecto fue desarrollado cumpliendo las siguientes restricciones académicas:

SIN Funciones Lambda:

En lugar de usar lambda, se definen funciones con nombre explícito:

```
# ❌ NO permitido en el proyecto
sorted(paises, key=lambda p: p['nombre'])

# ✅ Implementado en el proyecto
def obtener_nombre(pais):
    return pais['nombre']

sorted(paises, key=obtener_nombre)
```

SIN Manejo de Excepciones (try-except):

Las validaciones se realizan mediante condicionales y verificaciones manuales:

```
# ❌ NO permitido en el proyecto
try:
    numero = int(valor)
except ValueError:
    print("Error: valor inválido")

# ✅ Implementado en el proyecto
```

```
    if not valor.isdigit():
        print("Error: valor debe ser numérico")
        return False

    numero = int(valor)
```

SIN Clases (Programación Orientada a Objetos):

El proyecto utiliza únicamente funciones y módulos, sin definir clases.

SIN Recursividad:

Todos los algoritmos son iterativos, sin uso de funciones recursivas.

10.2. Validaciones Sin Excepciones

Estrategia de validación implementada:

1. Verificar tipo de dato manualmente:
    - Usar .isdigit() para números
    - Usar .strip() para detectar strings vacíos
2. Verificar rangos lógicos:
    - Población > 0
    - Superficie > 0
    - Min < Max en rangos
3. Retornar tuplas con resultado:
    - (exito, valor, mensaje_error)
4. Mensajes descriptivos de error:

- Indicar exactamente qué falló y por qué

REFERENCIAS BIBLIOGRÁFICAS

Documentación Oficial:

[1] Python Software Foundation. (2024). Python 3 Documentation.

Disponible en: https://docs.python.org/3/

[2] Python Software Foundation. (2024). csv — CSV File Reading and Writing.

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

[3] Python Software Foundation. (2024). Built-in Functions.

Disponible en: https://docs.python.org/3/library/functions.html

Libros de Referencia:

[4] Downey, A. B. (2015). Think Python: How to Think Like a Computer Scientist

(2nd ed.). O'Reilly Media.

[5] Matthes, E. (2019). Python Crash Course: A Hands-On, Project-Based

Introduction to Programming (2nd ed.). No Starch Press.

[6] Lutz, M. (2013). Learning Python (5th ed.). O'Reilly Media.

Recursos en Línea:

[7] Real Python. (2024). Python Tutorials.

Disponible en: https://realpython.com/

[8] W3Schools. (2024). Python Tutorial.

Disponible en: https://www.w3schools.com/python/

[9] GeeksforGeeks. (2024). Python Programming Language.

Disponible en: https://www.geeksforgeeks.org/python-programming-language/

[10] Sweigart, A. (2015). Automate the Boring Stuff with Python.

Disponible en: https://automatetheboringstuff.com/

Estándares Técnicos:

[11] Shafranovich, Y. (2005). RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files. Internet Engineering Task Force.

Disponible en: https://tools.ietf.org/html/rfc4180

Material Académico:

[12] Universidad Tecnológica Nacional. (2025). Programación 1 - Material de Cátedra.

Tecnicatura Universitaria en Programación a Distancia (TUPAD).

Diseño del caso práctico

DESCRIPCIÓN DEL SISTEMA

El sistema desarrollado es una aplicación de consola que permite gestionar un catálogo de países mediante operaciones CRUD, consultas, filtrados y análisis estadístico. La información se almacena en un archivo CSV que garantiza la persistencia de los datos entre ejecuciones.

ANÁLISIS DEL ENUNCIADO

La consigna solicitaba desarrollar un sistema con las siguientes funcionalidades mínimas:

- ✓ Agregar países con validación de datos
- ✓ Actualizar población y superficie de países existentes
- ✓ Buscar países por nombre (coincidencia exacta y parcial)
- ✓ Filtrar países por continente, rango de población y rango de superficie
- ✓ Ordenar países por nombre, población o superficie (ascendente/descendente)
- ✓ Mostrar estadísticas: máximos, mínimos, promedios y conteos
- ✓ Persistencia de datos en archivo CSV
- ✓ Validación exhaustiva de entradas

ESTRUCTURA DE CARPETAS Y ARCHIVOS

```
TP INTEGRADOR/
│
├── tp_integrador.py          ← Archivo principal con menú
│
├── modulos/                  ← Paquete de módulos
│   ├── __init__.py           ← Inicialización del paquete
│   ├── datos.py              ← CRUD y persistencia CSV
│   ├── busquedas.py          ← Búsqueda exacta y parcial
│   ├── estadisticas.py       ← Cálculos estadísticos
│   ├── filtros.py            ← Filtrado por criterios
│   ├── ordenamiento.py       ← Ordenamiento de datos
│   ├── utilidades.py         ← Formateo y presentación
│   └── validaciones.py       ← Validación de entradas
│
└── datos/
    └── paises.csv            ← Base de datos en CSV
```

DISEÑO DE ESTRUCTURAS DE DATOS

Estructura Principal: Lista de Diccionarios

```
paises = [
        {
            'nombre': 'Argentina',
            'poblacion': 45195774,
            'superficie': 2780400,
            'continente': 'America'
        },
        {
            'nombre': 'Brasil',
            'poblacion': 212559417,
            'superficie': 8515767,
```

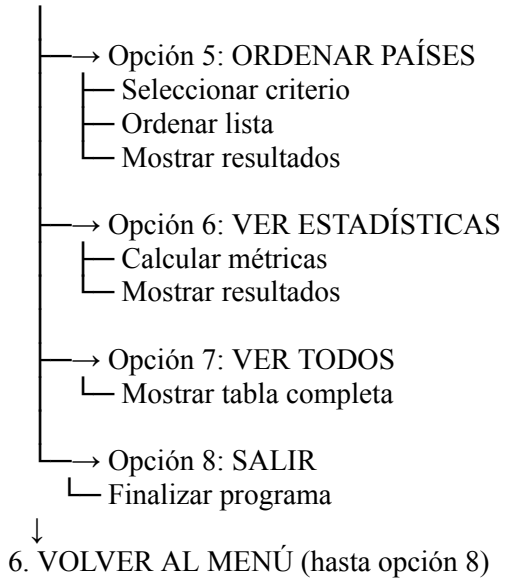
```
        'continente': 'America'
    }
]
```

Justificación:

- Lista: permite mantener orden y acceso por índice
- Diccionario: acceso descriptivo por clave (más legible)
- Combinación: ideal para datos tabulares

FLUJO DE OPERACIONES

1. INICIO DEL PROGRAMA
   ↓
2. CARGAR DATOS desde paises.csv
   ↓
3. MOSTRAR MENÚ (bucle while)
   ↓
4. USUARIO SELECCIONA OPCIÓN (1-8)
   ↓
5. EJECUTAR FUNCIONALIDAD
   - → Opción 1: AGREGAR PAÍS
     - Validar datos
     - Agregar a lista
     - Guardar en CSV
   - → Opción 2: ACTUALIZAR PAÍS
     - Buscar país
     - Modificar datos
     - Guardar en CSV
   - → Opción 3: BUSCAR PAÍS
     - Solicitar nombre
     - Buscar en lista
     - Mostrar resultado
   - → Opción 4: FILTRAR PAÍSES
     - Seleccionar criterio
     - Aplicar filtro
     - Mostrar resultados



↓

6. VOLVER AL MENÚ (hasta opción 8)

DESCRIPCIÓN DE MÓDULOS

1. datos.py - Gestión de Persistencia

- cargar_desde_csv(): Lee el archivo y carga los países en memoria
- guardar_en_csv(): Persiste los cambios en el archivo
- agregar_pais(): Añade un nuevo país al catálogo
- actualizar_pais(): Modifica población y superficie
- obtener_todos(): Retorna la lista completa de países
- buscar_por_nombre(): Localiza un país específico

2. busquedas.py - Localización de Datos

- buscar_exacto(): Búsqueda por nombre exacto (case-insensitive)
- buscar_parcial(): Búsqueda por coincidencia parcial

3. filtros.py - Filtrado de Datos

- filtrar_por_continente(): Filtra por continente específico

- filtrar_por_rango_poblacion(): Filtra por rango de habitantes
- filtrar_por_rango_superficie(): Filtra por rango de extensión

4. ordenamiento.py - Reorganización de Datos

- ordenar_por_nombre(): Orden alfabético (A→Z o Z→A)
- ordenar_por_poblacion(): Orden numérico por habitantes
- ordenar_por_superficie(): Orden numérico por extensión

5. estadisticas.py - Cálculos Estadísticos

- pais_mayor_poblacion(): Encuentra el país más poblado
- pais_menor_poblacion(): Encuentra el país menos poblado
- promedio_poblacion(): Calcula población promedio
- promedio_superficie(): Calcula superficie promedio
- contar_por_continente(): Cuenta países por continente

6. utilidades.py - Presentación

- mostrar_tabla_paises(): Presenta datos en formato tabular
- mostrar_pais_individual(): Muestra detalle de un país
- formatear_numero(): Agrega separadores de miles
- limpiar_pantalla(): Limpia la consola

7. validaciones.py - Validación de Entradas

- validar_numero_positivo(): Verifica números positivos
- validar_texto_no_vacio(): Verifica texto válido
- validar_rango(): Verifica que min < max

8. tp_integrador.py - Archivo Principal

- main(): Función principal del programa
- mostrar_menu(): Muestra opciones disponibles
- agregar_pais_menu(): Interfaz para agregar país
- actualizar_pais_menu(): Interfaz para actualizar
- buscar_pais_menu(): Interfaz para búsqueda
- filtrar_paises_menu(): Interfaz para filtrado
- ordenar_paises_menu(): Interfaz para ordenamiento
- ver_estadisticas_menu(): Interfaz para estadísticas

PRINCIPIOS DE DISEÑO APLICADOS

1. Separación de Responsabilidades (SoC)

Cada módulo tiene una única responsabilidad bien definida.

2. Don't Repeat Yourself (DRY)

Funciones reutilizables en lugar de código duplicado.

3. Modularidad

Módulos independientes que pueden modificarse sin afectar otros.

4. Validación Exhaustiva

Validación manual sin excepciones, cumpliendo restricciones académicas.

5. Interfaz Clara

Menú intuitivo con mensajes descriptivos para el usuario.

Metodología utilizada

ENFOQUE DE DESARROLLO

El desarrollo del proyecto se llevó a cabo en un período de 4 días, trabajando en equipo de 2 integrantes con reuniones virtuales para coordinar el trabajo y realizar integraciones del código.

Se adoptó un enfoque de desarrollo colaborativo donde cada integrante asumió responsabilidades específicas, manteniendo comunicación constante para asegurar la integración correcta de todas las partes del sistema.

HERRAMIENTAS Y TECNOLOGÍAS

Lenguaje de Programación:

• Python 3.x

• Módulos estándar: csv, os, sys

• Sin uso de librerías de terceros

Entorno de Desarrollo:

• Editor: Visual Studio Code (VSCode)

• Terminal: Para ejecución y pruebas del programa

• Sistema operativo: Multiplataforma (Linux/Mac)

Control de Versiones:

• Git: Para gestión de versiones del código

• GitHub: Para repositorio remoto y colaboración

• Ramas de trabajo: tp-alex y tp-cesia (una por integrante)

• Repositorio: https://github.com/alexdevep7/Trabajo-Practico-Integrador-P1

Herramientas de Documentación:

• Google Docs: Para elaboración del documento PDF

• Markdown: Para el README.md del repositorio

• Claude AI: Como asistente en el desarrollo y documentación

Comunicación del Equipo:

• Reuniones virtuales para coordinación (Google meet)

• Mensajería instantánea para consultas rápidas (Whatsapp)

• GitHub para revisión de código

PROCESO DE DESARROLLO

Día 1 - Análisis y Diseño:

• Lectura y análisis del enunciado del Trabajo Práctico

• Definición de la arquitectura modular (8 módulos)

• Diseño de estructuras de datos (lista de diccionarios)

• Planificación de la división del trabajo

• Creación del repositorio en GitHub

Día 2 - Implementación Base:

• Desarrollo del código principal por parte de uno de los integrantes

• Creación de los módulos fundamentales:

- datos.py (CRUD y persistencia)

- validaciones.py (validación de entradas)

- utilidades.py (presentación)

• Implementación del archivo principal con menú interactivo

• Pruebas iniciales de funcionalidad básica

Día 3 - Extensión y Refinamiento:

• Adición de bloques de código complementarios

• Implementación de módulos adicionales:

- busquedas.py (búsqueda exacta y parcial)

- filtros.py (filtrado por criterios)

- ordenamiento.py (ordenamiento de datos)

- estadisticas.py (cálculos estadísticos)

• Integración de todos los módulos

• Pruebas exhaustivas de cada funcionalidad

• Corrección de errores detectados

• Subida colaborativa del código a GitHub

Día 4 - Documentación y Entrega:

• Elaboración del documento PDF del proyecto

• Investigación y redacción del marco teórico

• Creación de capturas de pantalla del sistema funcionando

• Actualización del README.md en GitHub

• Preparación de diapositivas para presentación

• Revisión final y ajustes

ORGANIZACIÓN DEL TRABAJO EN EQUIPO

División de Responsabilidades:

Integrante 1 (Cesia):

• Desarrollo del código principal del sistema

• Implementación de la lógica core de los módulos

• Creación de diapositivas para la presentación

• Colaboración en la investigación del marco teórico

Integrante 2 (Alfredo):

• Adición de bloques de código complementarios

• Creación del repositorio en GitHub

• Elaboración del 80% del documento PDF

• Redacción del marco teórico y secciones técnicas

• Coordinación de la entrega final

Tareas Compartidas:

• Investigación de conceptos para el marco teórico

• Subida y actualización del código en GitHub

• Pruebas y validación del sistema

• Revisión y corrección de errores

• Documentación del README.md

DESAFÍOS ENFRENTADOS

Desafío Principal: Coordinación de Horarios

El mayor desafío fue la coordinación de reuniones de trabajo, ya que ambos integrantes del equipo tienen responsabilidades laborales, lo que dificultó encontrar horarios en común para sesiones sincrónicas de desarrollo.

Solución Implementada:

- Reuniones virtuales programadas con anticipación
- Trabajo asincrónico con comunicación constante
- Uso intensivo de GitHub para compartir avances
- Mensajería instantánea para consultas rápidas
- División clara de tareas para trabajo independiente

Desafíos Técnicos:

- Validación de datos sin uso de try-except (restricción académica)
- Implementación de funciones sin lambda (restricción académica)
- Integración correcta de los 8 módulos
- Persistencia consistente en archivo CSV

Soluciones Técnicas:

- Uso de métodos como .isdigit() para validación manual
- Definición de funciones con nombre explícito para ordenamiento
- Pruebas iterativas de cada módulo antes de integrar
- Revisión exhaustiva del código entre ambos integrantes

PRUEBAS Y VALIDACIÓN

Estrategia de Testing:

• Pruebas unitarias informales de cada función

• Pruebas de integración al combinar módulos

• Validación de casos de uso típicos

• Pruebas de casos extremos (datos inválidos, límites)

• Verificación de persistencia correcta en CSV

Casos de Prueba Ejecutados:

✓ Agregar países con datos válidos

✓ Agregar países con datos inválidos (validación)

✓ Buscar países existentes e inexistentes

✓ Filtrar por diferentes continentes

✓ Filtrar por rangos válidos e inválidos

✓ Ordenar por diferentes criterios

✓ Calcular estadísticas con diferentes tamaños de dataset

✓ Actualizar datos de países

✓ Persistencia tras cerrar y reabrir el programa

GESTIÓN DE VERSIONES CON GIT

Flujo de Trabajo:

1. Creación del repositorio remoto en GitHub

2. Clonación local del repositorio por ambos integrantes

3. Trabajo en ramas individuales:

- Rama tp-alex (Alfredo)
- Rama tp-cesia (Cesia)

4. Commits frecuentes con mensajes descriptivos
5. Push de cambios al repositorio remoto
6. Revisión de código entre integrantes
7. Integración de ambas ramas en la rama principal

Ejemplo de Flujo de Comandos:

```
# Alfredo trabajando en su rama
git checkout tp-alex
git add .
git commit -m "Actualizar README y corregir función duplicada"
git push origin tp-alex

# Cesia trabajando en su rama
git checkout tp-cesia
git add .
git commit -m "Implementar módulo de estadísticas"
git push origin tp-cesia
```

Esta estrategia de ramas separadas nos permitió que ambos trabajamos simultáneamente sin conflictos, facilitando el desarrollo paralelo de diferentes componentes del sistema.

LECCIONES APRENDIDAS

Aspectos Técnicos:

• La modularización facilita el mantenimiento y escalabilidad del código

• Las validaciones manuales requieren más código pero son más explícitas

• La persistencia en CSV es simple pero efectiva para proyectos pequeños

• La arquitectura bien planificada ahorra tiempo en la implementación

Aspectos de Trabajo en Equipo:

• La comunicación clara es fundamental para el trabajo colaborativo

• La división de tareas según fortalezas optimiza el tiempo

• El trabajo asincrónico es posible con buenas herramientas (GitHub)

• La planificación anticipada de reuniones es esencial cuando hay limitaciones de horario

• Git/GitHub es indispensable para trabajo colaborativo en código

ADAPTACIÓN A RESTRICCIONES ACADÉMICAS

El proyecto cumplió exitosamente con todas las restricciones establecidas:

✓ Sin funciones lambda:

Se definieron funciones con nombre explícito para todas las operaciones

✓ Sin try-except:

Se implementó validación manual usando condicionales y métodos como .isdigit()

✓ Sin clases:

Se utilizó programación funcional pura con módulos independientes

✓ Sin recursividad:

Todos los algoritmos se implementaron de forma iterativa

Estas restricciones, aunque inicialmente parecían limitantes, resultaron ser un ejercicio valioso para comprender los conceptos fundamentales de programación sin depender de abstracciones avanzadas.

Resultados obtenidos

FUNCIONALIDADES IMPLEMENTADAS

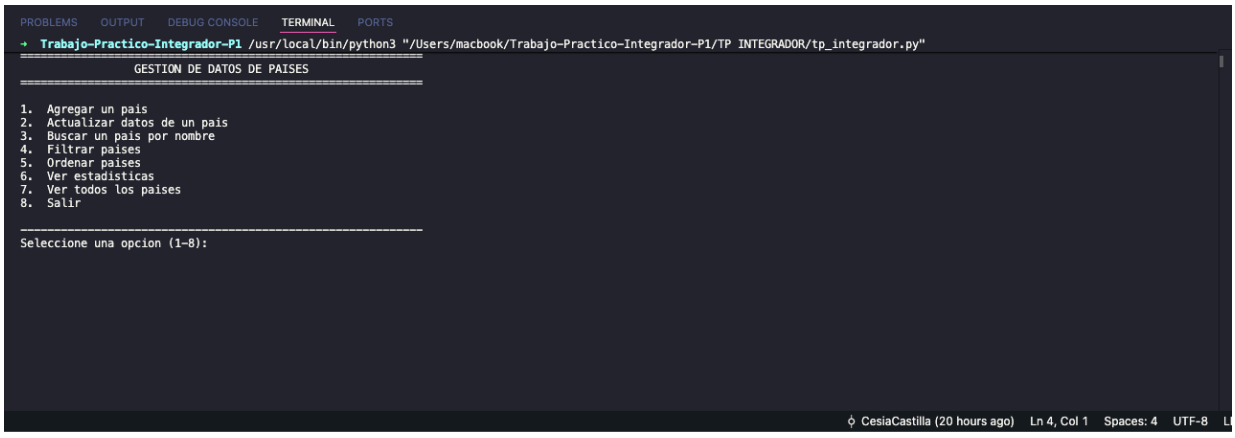
El sistema de gestión de países fue desarrollado exitosamente, cumpliendo con todas las funcionalidades requeridas por la consigna. A continuación se presenta la demostración visual del sistema en funcionamiento, con capturas de pantalla que evidencian cada una de las operaciones implementadas.

CAPTURAS DE PANTALLA DEL SISTEMA EN FUNCIONAMIENTO

6.1. Menú Principal

Al ejecutar el programa, el sistema presenta un menú interactivo con 8 opciones disponibles. Este menú es el punto de entrada para todas las funcionalidades del sistema y permanece visible hasta que el usuario selecciona la opción de salir.

Figura 1: Menú principal del sistema de gestión de países



6.2. Funcionalidad: Agregar País

6.2.1. Ingreso de Datos

Al seleccionar la opción 1 del menú, el sistema solicita al usuario que ingrese los datos del nuevo país: nombre, población, superficie y continente. Los datos son validados antes de ser almacenados.

Figura 2: Ingreso de datos para agregar un nuevo país

6.2.2. Confirmación de Operación Exitosa

Una vez que los datos han sido validados correctamente, el sistema agrega el país al catálogo y guarda los cambios en el archivo CSV automáticamente, mostrando un mensaje de confirmación al usuario.

Figura 3: Confirmación de país agregado exitosamente

6.2.3. Validación de Datos Inválidos

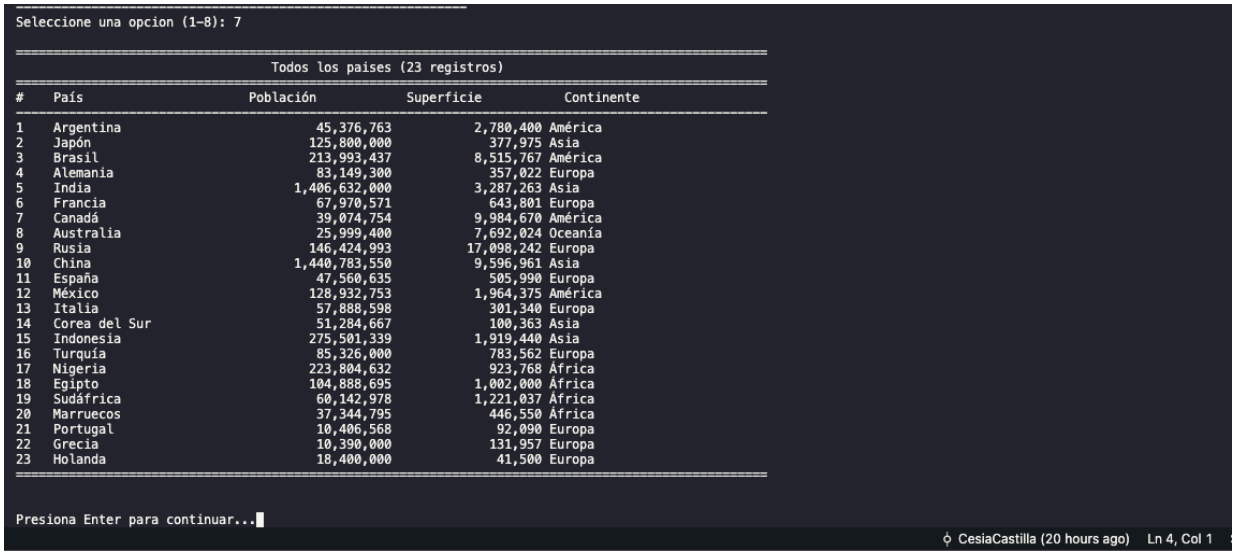
El sistema implementa validaciones exhaustivas. Cuando el usuario ingresa datos inválidos (campos vacíos, números negativos, etc.), el sistema muestra mensajes de error descriptivos y no permite continuar hasta que los datos sean correctos.

Figura 4: Validación de datos inválidos con mensaje de error

6.3. Funcionalidad: Ver Todos los Países

Al seleccionar la opción 7, el sistema presenta todos los países del catálogo en formato tabular, con columnas claramente definidas para nombre, población, superficie y continente. Los números se formatean con separadores de miles para mejor legibilidad.

Figura 5: Vista de todos los países en formato tabular



6.4. Funcionalidad: Búsqueda de Países

6.4.1. Ingreso de Búsqueda

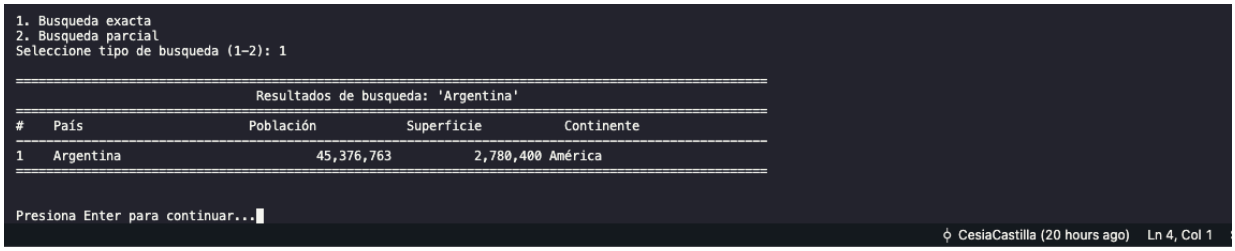
Al seleccionar la opción 3, el sistema solicita el nombre del país a buscar y ofrece dos tipos de búsqueda: exacta o parcial. Esto permite flexibilidad en la localización de países.

Figura 6: Ingreso de criterio de búsqueda

6.4.2. Resultado de Búsqueda Exitosa

Cuando el sistema encuentra el país buscado, lo presenta en formato tabular con todos sus datos. La búsqueda es case-insensitive, permitiendo buscar sin preocuparse por mayúsculas o minúsculas.

Figura 7: Resultado de búsqueda exitosa



6.4.3. País No Encontrado

Cuando el país buscado no existe en el catálogo, el sistema muestra un mensaje claro informando que no se encontraron resultados, evitando confusión al usuario.

Figura 8: Mensaje cuando el país buscado no existe

6.5. Funcionalidad: Filtrado de Países

6.5.1. Filtrado por Continente - Entrada

Al seleccionar la opción 4, el sistema ofrece tres tipos de filtros. En este caso, el filtrado por continente permite al usuario especificar el continente del cual desea ver los países.

Figura 9: Selección de criterio de filtrado por continente

6.5.2. Filtrado por Continente - Resultado

El sistema muestra todos los países que pertenecen al continente especificado, presentados en formato tabular. Esto permite análisis específicos por región geográfica.

Figura 10: Resultado del filtrado por continente

```
Continentes disponibles: América, Asia, Europa, Oceanía, África
Continente: Asia

================================================================================
                              Resultados del filtro
================================================================================
#    País                 Población           Superficie          Continente
--------------------------------------------------------------------------------
1    Japón                  125,800,000            377,975 Asia
2    India                1,406,632,000          3,287,263 Asia
3    China                1,440,783,550          9,596,961 Asia
4    Corea del Sur           51,284,667            100,363 Asia
5    Indonesia              275,501,339          1,919,440 Asia
================================================================================

Presiona Enter para continuar...
                                                CesiaCastilla (20 hours ago)  Ln 4, Col 1
```

6.5.3. Filtrado por Rango de Población - Entrada

El sistema también permite filtrar países por rango de población, solicitando al usuario los valores mínimo y máximo. Se valida que el rango sea lógico (mínimo < máximo).

Figura 11: Ingreso de rango de población para filtrado

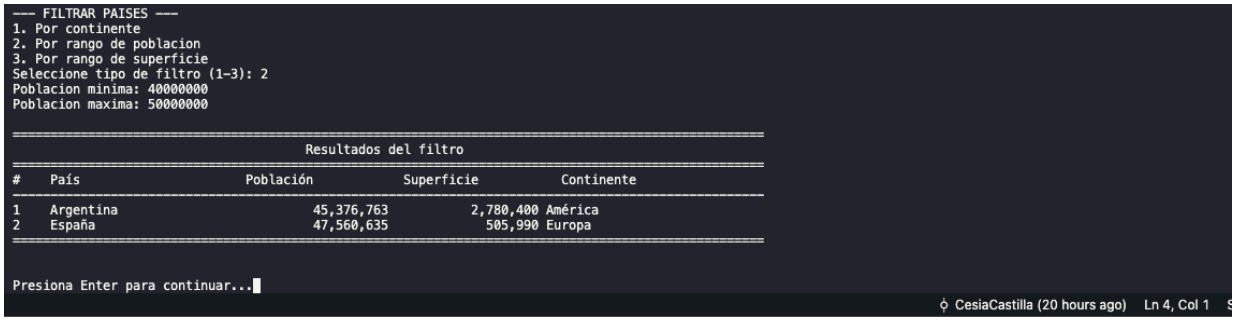
```
------------------------------------------------------
Seleccione una opcion (1-8): 4

--- FILTRAR PAISES ---
1. Por continente
2. Por rango de poblacion
3. Por rango de superficie
Seleccione tipo de filtro (1-3): 2
Poblacion minima: 40000000
Poblacion maxima: 50000000
```

6.5.4. Filtrado por Rango de Población - Resultado

El sistema muestra únicamente los países cuya población se encuentra dentro del rango especificado, facilitando análisis demográficos específicos.

Figura 12: Resultado del filtrado por rango de población

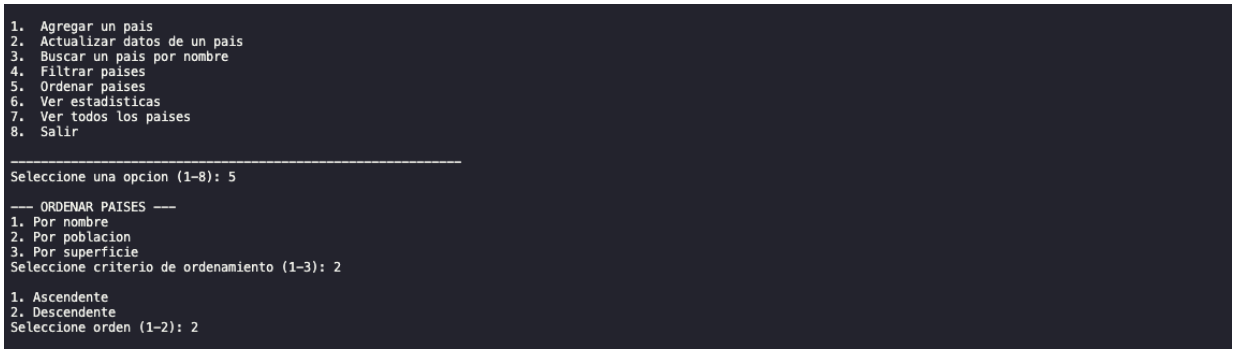


6.6. Funcionalidad: Ordenamiento de Países

6.6.1. Selección de Criterio de Ordenamiento

Al seleccionar la opción 5, el sistema permite ordenar los países por tres criterios diferentes: nombre, población o superficie. Además, permite elegir entre orden ascendente o descendente.

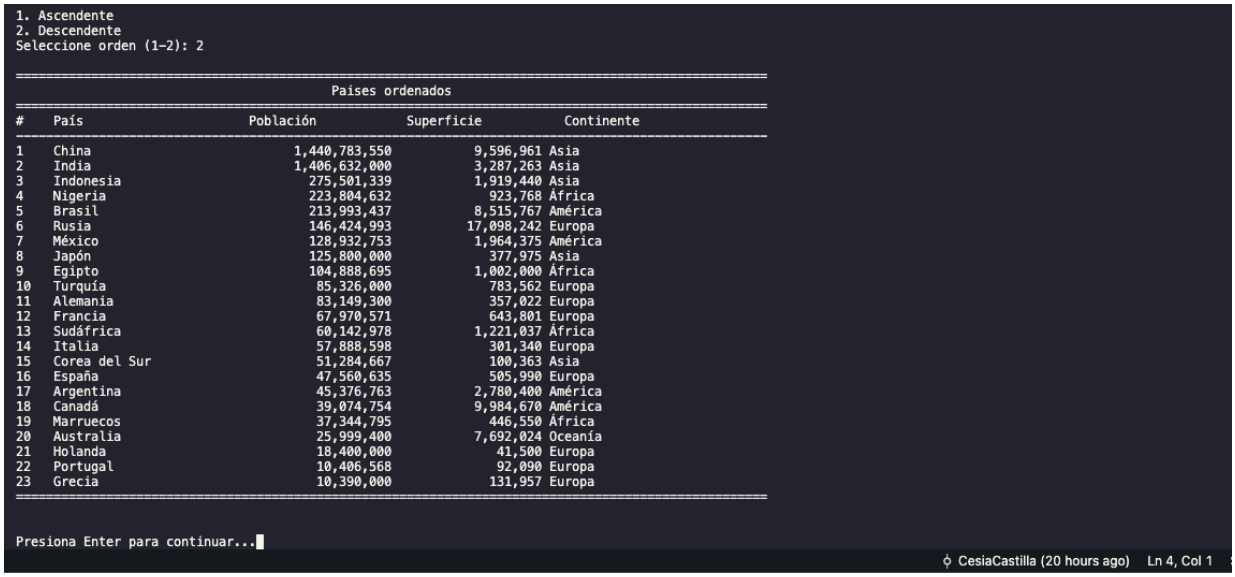
Figura 13: Selección de criterio y orden para ordenamiento



6.6.2. Resultado del Ordenamiento

El sistema presenta los países ordenados según el criterio seleccionado. En este ejemplo, los países están ordenados por población de mayor a menor, lo que permite identificar rápidamente los países más poblados.

Figura 14: Países ordenados según criterio seleccionado



6.7. Funcionalidad: Estadísticas del Sistema

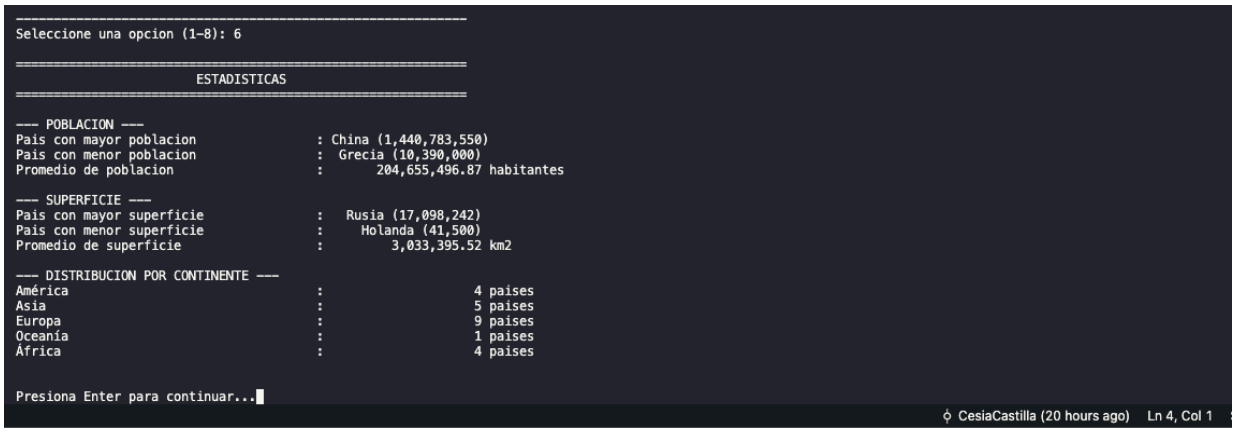
Al seleccionar la opción 6, el sistema calcula y presenta estadísticas completas del catálogo de países:

- Población: País con mayor y menor población, promedio poblacional
- Superficie: País con mayor y menor superficie, promedio de superficie

• Distribución: Cantidad de países por continente

Esta funcionalidad permite obtener una visión general del dataset de forma rápida y clara.

Figura 15: Estadísticas completas del catálogo de países



6.8. Funcionalidad: Actualizar Datos de País

6.8.1. Ingreso de Datos a Actualizar

Al seleccionar la opción 2, el sistema permite actualizar la población y/o superficie de un país existente.

Primero busca el país por nombre y muestra sus datos actuales, luego solicita los nuevos valores.

Figura 16: Actualización de datos de un país existente

6.8.2. Confirmación de Actualización

Una vez actualizados los datos, el sistema muestra un mensaje de confirmación indicando qué campos fueron modificados y guarda automáticamente los cambios en el archivo CSV.

Figura 17: Confirmación de actualización exitosa

EVALUACIÓN DEL DESARROLLO

Aspectos Exitosos

1. Arquitectura Modular

La división del código en 8 módulos especializados facilitó enormemente el desarrollo y el mantenimiento. Cada módulo tiene una responsabilidad clara y puede ser modificado sin afectar a los demás.

2. Validación Robusta

La implementación de validaciones manuales sin uso de try-except resultó ser efectiva. El uso de métodos como .isdigit() y condicionales permitió crear un sistema de validación claro y predecible.

3. Interfaz de Usuario Intuitiva

El menú interactivo resultó ser claro y fácil de usar. Los mensajes descriptivos guían al usuario a través de cada operación sin generar confusión.

4. Persistencia Automática

La integración entre las operaciones de modificación y el guardado automático en CSV garantiza que los datos siempre estén sincronizados, evitando pérdida de información.

5. Formato de Presentación

El uso de tablas con formato ASCII y separadores de miles en los números hace que los datos sean fáciles de leer y profesionales en apariencia.

Desafíos Enfrentados

1. Coordinación de Horarios

El mayor desafío fue la coordinación de reuniones entre los integrantes del equipo, ya que ambos trabajamos y encontrar horarios compatibles fue complicado.

Solución: Se establecieron reuniones programadas con anticipación y se adoptó un modelo de trabajo asincrónico apoyado en GitHub para compartir avances.

2. Validación sin Excepciones

Implementar validaciones robustas sin poder usar try-except requirió pensar cuidadosamente en todos los casos posibles de error.

Solución: Se creó un módulo especializado (validaciones.py) con funciones que retornan tuplas indicando el resultado de la validación, el valor procesado y mensajes de error.

3. Integración de Módulos

Asegurar que todos los módulos funcionaran correctamente juntos requirió pruebas exhaustivas.

Solución: Se probó cada módulo de forma independiente antes de integrarlo, y se realizaron pruebas de integración progresivas.

4. Funciones sin Lambda

No poder usar funciones lambda para ordenamiento obligó a definir funciones con nombre explícito para cada criterio.

Solución: Se crearon funciones extractoras claras (obtener_nombre, obtener_poblacion, etc.) que resultaron ser más legibles que lambdas.

Decisiones de Diseño que Cambiaron Durante el Desarrollo

1. Estructura de Datos Inicial

Inicialmente se consideró usar diccionarios separados para cada atributo, pero se optó por una lista de diccionarios que resultó más natural y fácil de manejar.

2. Organización de Módulos

Durante el desarrollo se identificó la necesidad de separar utilidades.py y validaciones.py, que originalmente estaban en un solo módulo. Esta separación mejoró la claridad del código.

3. Formato de Presentación

Se experimentó con diferentes formatos para mostrar los datos (listas simples, tablas básicas) hasta llegar al formato actual con box-drawing que ofrece mejor legibilidad.

Comunicación del Equipo

La comunicación entre nosotros fue efectiva a pesar de las limitaciones de horario:

• Reuniones virtuales programadas: Se establecieron horarios fijos para sesiones de trabajo conjunto
• Mensajería instantánea: Permitió resolver dudas rápidas sin necesidad de reuniones formales
• GitHub como punto de encuentro: El repositorio sirvió como espacio central donde ambos podían ver y revisar el trabajo del otro
• División clara de tareas: Cada integrante sabía exactamente qué debía hacer, minimizando duplicación de esfuerzos

La experiencia de trabajo en equipo fue positiva y permitió completar el proyecto exitosamente en el tiempo establecido.

CUMPLIMIENTO DE REQUISITOS

El sistema cumple con todos los requisitos establecidos en la consigna:

✓ Agregar países con validación completa

✓ Actualizar población y superficie

✓ Búsqueda exacta y parcial por nombre

✓ Filtrado por continente, población y superficie

✓ Ordenamiento por múltiples criterios (asc/desc)

✓ Estadísticas completas (máx, mín, promedios, conteos)

✓ Persistencia en archivo CSV

✓ Validación exhaustiva de entradas

✓ Código modular y bien organizado

✓ Sin lambdas, sin try-except, sin clases, sin recursividad

El resultado final es un sistema funcional, robusto y bien documentado que demuestra la aplicación práctica de todos los conceptos vistos en la materia Programación 1.

Conclusiones

APRENDIZAJES TÉCNICOS

El desarrollo de este Trabajo Práctico Integrador permitió consolidar y aplicar de manera práctica los conceptos fundamentales de programación vistos durante la cursada de Programación 1.

Estructuras de Datos:

El uso combinado de listas y diccionarios demostró ser una solución elegante y eficiente para representar datos tabulares. La lista de diccionarios permite:

- Mantener un orden lógico de los elementos
- Acceder a los datos de forma descriptiva mediante claves
- Iterar fácilmente sobre toda la colección
- Realizar operaciones de búsqueda, filtrado y ordenamiento de manera natural

Esta estructura de datos es fundamental en el desarrollo de software y se aplica en múltiples contextos, desde aplicaciones web hasta análisis de datos.

Modularización y Funciones:

La división del código en 8 módulos especializados fue uno de los aprendizajes más valiosos del proyecto. La modularización ofrece beneficios concretos:

- Facilita el mantenimiento: los cambios se localizan en módulos específicos
- Mejora la legibilidad: cada módulo tiene un propósito claro
- Permite el trabajo colaborativo: cada integrante puede trabajar en módulos diferentes
- Favorece la reutilización: las funciones pueden usarse en múltiples contextos

Aplicar el principio "una función = una responsabilidad" resultó en un código limpio, organizado y fácil de entender, incluso al revisar el código semanas después de haberlo escrito.

Validación de Datos Sin Excepciones:

Implementar validaciones robustas sin poder usar try-except fue inicialmente un desafío, pero resultó ser un ejercicio valioso que obligó a:

- Pensar cuidadosamente en todos los casos posibles de error
- Utilizar métodos como .isdigit() y condicionales de forma efectiva
- Diseñar funciones que retornen información completa sobre el resultado de la validación
- Crear mensajes de error descriptivos que ayuden al usuario

Esta experiencia demostró que las excepciones no son la única forma de manejar errores, y que la validación explícita puede ser igualmente efectiva y más predecible.

Persistencia con CSV:

El trabajo con archivos CSV enseñó conceptos importantes sobre persistencia de datos:

- Cómo estructurar datos para almacenamiento permanente
- La importancia de la consistencia en el formato de datos
- Técnicas de lectura y escritura de archivos
- Manejo de rutas y verificación de existencia de archivos

Aunque CSV es un formato simple, sirvió perfectamente para las necesidades del proyecto y demostró que no siempre se necesitan soluciones complejas para resolver problemas de almacenamiento de datos.

Algoritmos de Búsqueda y Ordenamiento:

La implementación de algoritmos básicos de búsqueda lineal y el uso de la función sorted() con funciones extractoras permitió comprender:

- Las diferencias entre búsqueda exacta y parcial
- Cómo optimizar comparaciones con .lower() para case-insensitivity
- El funcionamiento de algoritmos de ordenamiento
- La importancia de la complejidad temporal en el rendimiento

Restricciones Académicas como Oportunidad de Aprendizaje:

Las restricciones establecidas (sin lambdas, sin try-except, sin clases, sin recursividad) no fueron limitaciones sino oportunidades para comprender los fundamentos:

- Definir funciones con nombre explícito mejoró la legibilidad del código
- Validaciones manuales obligaron a pensar en el control de flujo
- Programación funcional demostró que no siempre se necesitan clases
- Algoritmos iterativos son más directos y fáciles de seguir que recursivos

Estas restricciones aseguraron que el enfoque estuviera en los conceptos fundamentales, sin depender de abstracciones avanzadas.

IMPORTANCIA DE LAS HERRAMIENTAS UTILIZADAS

Python como Lenguaje de Aprendizaje

Python demostró ser una excelente elección para el aprendizaje de programación:

- Sintaxis clara y legible que facilita la comprensión del código

- Tipado dinámico que permite enfocarse en la lógica sin preocuparse por declaraciones de tipos
- Biblioteca estándar completa que cubre necesidades básicas sin dependencias externas
- Comunidad amplia con abundante documentación y recursos de aprendizaje

Visual Studio Code como Entorno de Desarrollo:

VSCode facilitó significativamente el proceso de desarrollo:

- Resaltado de sintaxis que mejora la lectura del código
- Autocompletado inteligente que acelera la escritura
- Terminal integrada que permite ejecutar y probar rápidamente
- Detección de errores en tiempo real que ayuda a identificar problemas temprano

Git y GitHub para Control de Versiones:

El uso de Git y GitHub fue fundamental para el trabajo colaborativo:

- Historial completo de cambios que permite rastrear la evolución del código
- Ramas independientes (tp-alex y tp-cesia) que permiten trabajo paralelo sin conflictos
- Repositorio remoto que funciona como respaldo y punto de sincronización
- Revisión de código entre integrantes antes de integrar cambios

Esta experiencia con control de versiones es invaluable, ya que Git es el estándar de la industria y será utilizado en cualquier entorno profesional de desarrollo de software.

Google Docs para Documentación Colaborativa:

La elaboración del documento en Google Docs permitió:

- Trabajo simultáneo en diferentes secciones
- Historial de versiones y posibilidad de deshacer cambios
- Acceso desde cualquier dispositivo con conexión a internet
- Comentarios y sugerencias para revisar el contenido antes de finalizarlo

Claude AI como Asistente de Desarrollo

El uso de Claude AI como herramienta de apoyo demostró el potencial de las herramientas de IA en el aprendizaje:

- Explicaciones detalladas de conceptos complejos
- Sugerencias de mejoras en la estructura del código
- Ayuda en la documentación y creación de contenido técnico
- Resolución de dudas puntuales durante el desarrollo

Es importante destacar que Claude fue utilizado como asistente, no como reemplazo del aprendizaje. Cada concepto fue comprendido y el código fue escrito y revisado manualmente por nosotros.

JUSTIFICACIÓN DEL USO DE HERRAMIENTAS

Cada herramienta seleccionada cumplió un propósito específico en el proyecto:

Python:

- Elegido por ser el lenguaje de la materia y por su claridad sintáctica
- Permite enfocarse en los conceptos de programación sin complejidad innecesaria
- Ampliamente usado en la industria para múltiples propósitos

Módulos Estándar (csv, os, sys):

- Evitan dependencias externas, facilitando la portabilidad del código
- Están bien documentados y son estables
- Suficientes para las necesidades del proyecto

VSCode:

- Gratuito y multiplataforma
- Extensible mediante plugins si se necesitara en el futuro
- Ampliamente utilizado en la industria

Git/GitHub:

- Estándar de la industria para control de versiones
- Esencial para trabajo colaborativo
- Habilidad transferible a cualquier proyecto futuro

Google Docs:

- Accesible sin instalación de software
- Familiar para ambos integrantes del equipo
- Facilita la colaboración en tiempo real

ORGANIZACIÓN DEL TRABAJO EN EQUIPO

El trabajo colaborativo entre nosotros fue un componente clave del éxito del proyecto.

División de Tareas Según Fortalezas

La asignación de responsabilidades se basó en las fortalezas de cada integrante:

- Cesia asumió el desarrollo del código core del sistema
- Alfredo se enfocó en la documentación técnica y la integración
- Ambos participaron en la revisión y pruebas del código

Esta división permitió optimizar el tiempo y aprovechar las habilidades de cada integrante.

Comunicación Efectiva

A pesar del desafío de coordinar horarios por compromisos laborales, se logró mantener comunicación efectiva mediante:

- Reuniones programadas con anticipación
- Mensajería instantánea para consultas rápidas
- Uso de GitHub para compartir avances y revisar código
- Documentación clara de lo realizado por cada integrante

La experiencia demostró que el trabajo remoto es viable cuando se utilizan las herramientas adecuadas y se mantiene comunicación constante.

Trabajo Asincrónico con GitHub:

El uso de ramas separadas (tp-alex y tp-cesia) permitió:

- Trabajo independiente sin esperar al otro integrante
- Desarrollo paralelo de diferentes componentes
- Integración ordenada mediante revisión de cambios
- Minimización de conflictos en el código

Esta metodología es similar a la utilizada en equipos de desarrollo profesionales.

Aprendizajes sobre Trabajo en Equipo:

La experiencia de trabajo colaborativo enseñó lecciones valiosas:

- La planificación anticipada ahorra tiempo durante la ejecución
- La comunicación clara es más importante que la cantidad de comunicación
- Documentar decisiones evita malentendidos
- Respetar los tiempos y compromisos del otro es fundamental
- La revisión mutua del código mejora la calidad del producto final

Estas habilidades blandas son tan importantes como las habilidades técnicas en el desarrollo de software profesional.

APLICABILIDAD DE LOS CONOCIMIENTOS ADQUIRIDOS

Los conceptos y técnicas aprendidas en este proyecto tienen aplicaciones directas en contextos reales:

Desarrollo de Aplicaciones:

- Cualquier aplicación que gestione datos (inventarios, clientes, productos) utilizará estructuras similares
- La modularización es fundamental en proyectos de mayor escala
- La validación de datos es crítica en aplicaciones que interactúan con usuarios

Análisis de Datos:

- Las técnicas de filtrado, ordenamiento y cálculo de estadísticas son básicas en análisis de datos
- El trabajo con archivos CSV es común en ciencia de datos
- Python es uno de los lenguajes más utilizados en este campo

Desarrollo Web:

- Los conceptos de estructuras de datos se aplican al manejo de bases de datos
- La arquitectura modular es esencial en aplicaciones web
- La persistencia de datos es fundamental en cualquier aplicación con usuarios

Automatización de Tareas:

- Python es excelente para automatización
- El manejo de archivos aprendido se aplica a procesamiento de datos en batch
- Las validaciones son importantes en scripts que procesan información crítica

Trabajo Colaborativo:

- Git/GitHub es universal en desarrollo de software
- La capacidad de trabajar en equipo es requerida en la mayoría de posiciones
- La documentación clara es apreciada en cualquier organización

REFLEXIÓN FINAL

El Trabajo Práctico Integrador cumplió con su objetivo principal: afianzar los conocimientos que fuimos adquiriendo durante la cursada de Programación 1.

Más allá de haber implementado correctamente las funcionalidades requeridas, lo más valioso del proyecto fue todo lo que aprendimos en el proceso:

- Entender la importancia de la modularización y del código limpio

- Vivir de cerca el ciclo completo de desarrollo de software, desde el análisis hasta la documentación
- Trabajar en equipo, aprendiendo a coordinarnos y comunicarnos de forma efectiva
- Poner en práctica los conceptos teóricos en un proyecto real y funcional
- Desarrollar nuestro pensamiento lógico y la capacidad de resolver problemas

Este proyecto nos permitió confirmar que los fundamentos de programación que aprendimos son sólidos y totalmente aplicables a situaciones reales. Las restricciones académicas no fueron un obstáculo, sino una oportunidad para profundizar en los conceptos básicos que servirán como base para seguir creciendo en las próximas materias de la Tecnicatura.

Además, la experiencia de trabajo colaborativo fue clave: nos ayudó a organizarnos, repartir tareas y tomar decisiones conjuntas, algo fundamental en el desarrollo profesional de software.

En definitiva, este Trabajo Práctico Integrador no representa un final, sino un punto de partida. Nos deja la satisfacción de haber superado un desafío exigente y la motivación de seguir aprendiendo.

Con dedicación, organización y trabajo en equipo, demostramos que es posible crear software funcional, de calidad y con bases sólidas.

Referencias bibliográficas

DOCUMENTACIÓN OFICIAL

[1] Python Software Foundation. (2024). Python 3 Documentation.
Disponible en: https://docs.python.org/3/

[2] Python Software Foundation. (2024). csv — CSV File Reading and Writing.
Disponible en: https://docs.python.org/3/library/csv.html

[3] Python Software Foundation. (2024). Built-in Functions.
Disponible en: https://docs.python.org/3/library/functions.html

[4] Python Software Foundation. (2024). os — Miscellaneous operating system interfaces.
Disponible en: https://docs.python.org/3/library/os.html

LIBROS DE REFERENCIA

[5] Downey, A. B. (2015). Think Python: How to Think Like a Computer Scientist
(2nd ed.). O'Reilly Media.

[6] Matthes, E. (2019). Python Crash Course: A Hands-On, Project-Based
Introduction to Programming (2nd ed.). No Starch Press.

[7] Lutz, M. (2013). Learning Python (5th ed.). O'Reilly Media.

RECURSOS EN LÍNEA

[8] Real Python. (2024). Python Tutorials.
Disponible en: https://realpython.com/

[9] W3Schools. (2024). Python Tutorial.
Disponible en: https://www.w3schools.com/python/

[10] GeeksforGeeks. (2024). Python Programming Language.
Disponible en: https://www.geeksforgeeks.org/python-programming-language/

[11] Sweigart, A. (2015). Automate the Boring Stuff with Python.
Disponible en: https://automatetheboringstuff.com/

ESTÁNDARES TÉCNICOS

[12] Shafranovich, Y. (2005). RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files. Internet Engineering Task Force.
Disponible en: https://tools.ietf.org/html/rfc4180

MATERIAL ACADÉMICO

[13] Universidad Tecnológica Nacional. (2025). Programación 1 - Material de Cátedra. Tecnicatura Universitaria en Programación a Distancia (TUPAD).

[14] Universidad Tecnológica Nacional. (2025). Trabajo Práctico Integrador - Gestión de Datos de Países en Python. Consigna oficial.

REPOSITORIO DEL PROYECTO

[15] Castillo Belisario, A. & Cesia. (2025). Trabajo Práctico Integrador - Sistema de Gestión de Países. GitHub. Disponible en: https://github.com/alexdevep7/Trabajo-Practico-Integrador-P1