Informe Teórico - Trabajo Práctico Integrador

Gestión de Datos de Países en Python

Alumnos: Franco Gasparin D'Allotta (comision-5), Lautaro German Lugones (comision-7)
Docente: Cinthia Rigoni
Materia: Programación 1
Fecha de entrega: 11 de noviembre 2025

Listas

Introducción

Una lista es una estructura de datos que organiza y almacena elementos de manera secuencial y ordenada. Las listas permiten almacenar múltiples valores bajo un solo nombre y acceder a ellos mediante índices numéricos que comienzan desde cero.

Funciones y utilidad en programación

- **Organización:** Las listas ayudan a organizar la información de manera estructurada, permitiendo almacenar elementos en un orden específico.
- **Acceso rápido:** Permiten acceder a cualquier elemento mediante su índice en tiempo constante O(1).
- **Almacenamiento flexible:** Las listas pueden contener diferentes tipos de datos (números, strings, booleanos, otras listas).
- **Facilitan tareas repetitivas:** Permiten iterar sobre múltiples elementos de manera eficiente usando bucles.

Creación y operaciones básicas

python
```
# Declaración de una lista
lista_elementos = [1, 2, 3, 4, 5]
lista_strings = ["Ana", "Luis", "Pedro"]

# Lista vacía
nombre_lista = []

# Lista mixta
lista_mixta = [1, "Hola", 3.14, True]

# Operaciones principales
lista.append(elemento)       # Añadir elemento
lista.remove(elemento)       # Eliminar elemento
lista[indice] = nuevo_valor  # Actualizar elemento
lista1 + lista2              # Concatenar listas
elemento in lista            # Verificar existencia
```

Relación con nuestro proyecto

En nuestro sistema de gestión de países, utilizamos una **lista principal** llamada `catalogo` que almacena todos los países del sistema:

python
```
catalogo = []  # Lista principal que contiene todos los países
```

Esta lista es fundamental porque:

- Almacena todos los diccionarios de países de forma ordenada
- Permite agregar nuevos países dinámicamente con `catalogo.append(pais)`
- Facilita el recorrido de todos los países mediante bucles `for`
- En las funciones de búsqueda y filtrado, creamos listas temporales como `encontrados = []` y `filtrados = []` para almacenar resultados
- La función `mostrar_estadisticas()` recorre la lista completa para calcular promedios y encontrar valores máximos/mínimos

Diccionarios

Introducción

Los diccionarios son estructuras de datos que almacenan información en pares clave-valor (key-value). A diferencia de las listas, el acceso a los elementos no se realiza mediante índices numéricos sino a través de claves únicas. Cada clave debe ser única e inmutable (strings, números, tuplas), mientras que los valores pueden ser de cualquier tipo.

Funciones y utilidad en programación

- **Acceso rápido a datos:** Permiten recuperar un valor casi instantáneamente usando su clave asociada, con complejidad O(1).
- **Organización de datos relacionados:** Ideales para agrupar información estructurada, como los atributos de un objeto.
- **Creación de índices:** Funcionan como índices para búsquedas rápidas en grandes volúmenes de datos.
- **Contadores:** Útiles para contar frecuencias de elementos.
- **Estructuras anidadas:** Pueden contener otros diccionarios como valores, creando jerarquías complejas.

Creación y operaciones básicas

python
```
# Creación de diccionario
nombre_diccionario = {"key1": value1, "key2": value2}

# Operaciones principales
diccionario.values()          # Obtener todos los valores
diccionario.keys()            # Obtener todas las claves
diccionario["key"]            # Acceder a un valor
diccionario["key"] = nuevo    # Modificar o agregar par clave-valor
```

Relación con nuestro proyecto

En nuestro sistema, cada país está representado como un **diccionario** con cuatro claves específicas:

python
```
pais = {
   "nombre": nombre,
   "poblacion": int(poblacion),
   "superficie": int(superficie),
```

```
    "continente": continente
}
```

Los diccionarios son esenciales en el proyecto porque:

- Modelan perfectamente la estructura de datos de un país con atributos claramente identificados
- Permiten acceso directo a atributos específicos: `pais["nombre"]`, `pais["poblacion"]`
- En `actualizar_pais()`, modificamos valores específicos: `pais["poblacion"] = int(nueva_poblacion)`
- En `ordenar_paises()`, usamos las claves para ordenar: `key=lambda x: x["poblacion"]`
- En `mostrar_estadisticas()`, la variable `continentes = {}` es un diccionario que cuenta países por continente, usando el continente como clave y la cantidad como valor

Funciones

Introducción

Una función es un bloque de código reutilizable que realiza una tarea específica. Las funciones permiten dividir programas complejos en partes más pequeñas y manejables, siguiendo el principio de modularidad.

Componentes de una función

- **Entrada (argumentos/parámetros):** Datos que recibe la función
- **Proceso:** Instrucciones que ejecuta
- **Salida (retorno):** Resultado que devuelve (opcional)

Funciones y utilidad en programación

- **Modularidad:** Descomponen programas grandes en tareas específicas y manejables.
- **Reutilización de código:** Permiten escribir código una vez y usarlo múltiples veces, evitando duplicación.
- **Legibilidad:** Nombres descriptivos hacen el código más comprensible.
- **Abstracción:** Ocultan detalles de implementación complejos.
- **Mantenimiento:** Facilitan actualizaciones localizadas sin afectar el resto del programa.

Creación y tipos de funciones

```
python
# Función definida por el usuario
def nombre_funcion(argumentos):
    # Código a ejecutar
```

```
    return valor_retorno

# Funciones integradas (built-in)
print(), len(), input(), range()
```

Parámetros y ámbitos

- **Por valor:** Con tipos inmutables (int, str, tuple), los cambios internos no afectan la variable externa
- **Por referencia:** Con tipos mutables (list, dict, set), los cambios internos sí modifican el objeto original
- **Ámbito local:** Variables declaradas dentro de una función, visibles solo dentro de ella
- **Ámbito global:** Variables visibles para todas las funciones definidas después

Relación con nuestro proyecto

El proyecto está completamente **modularizado mediante funciones**, cada una con una responsabilidad específica:

```
python
def agregar_pais(catalogo):        # Agregar nuevo país
def actualizar_pais(catalogo):     # Actualizar datos existentes
def buscar_pais(catalogo):         # Búsqueda por nombre
def filtrar_paises(catalogo):      # Filtrar por criterios
def ordenar_paises(catalogo):      # Ordenar por campos
def mostrar_estadisticas(catalogo):# Calcular estadísticas
def mostrar_todos(catalogo):       # Listar países
def menu_principal():              # Controlar flujo del programa
```

Características importantes:

- Cada función recibe el `catalogo` como parámetro (paso por referencia)
- Las modificaciones al catálogo (append, sort) afectan la lista original
- Variables locales como `encontrados`, `filtrados`, `termino` solo existen dentro de sus funciones
- El código es más legible y mantenible: si necesitamos modificar cómo se agregan países, solo editamos `agregar_pais()`

Condicionales

Introducción

Las estructuras condicionales permiten que un programa tome decisiones y ejecute diferentes bloques de código según se cumplan o no ciertas condiciones. Son fundamentales para crear programas que respondan dinámicamente a diferentes situaciones.

Funciones y utilidad en programación

- **Control de flujo:** Permiten alterar el flujo de ejecución del programa según condiciones lógicas

- **Validación de datos:** Verifican que las entradas del usuario cumplan requisitos específicos
- **Toma de decisiones:** Ejecutan código diferente según el contexto
- **Manejo de errores:** Detectan situaciones anómalas y responden apropiadamente

Estructuras básicas

python
```
# Condicional simple
if condicion:
    # Código si es verdadero

# Condicional con alternativa
if condicion:
    # Código si es verdadero
else:
    # Código si es falso

# Condicionales múltiples
if condicion1:
    # Código 1
elif condicion2:
    # Código 2
else:
    # Código por defecto
```

Relación con nuestro proyecto

Los condicionales son utilizados extensivamente en todo el proyecto:

Validación de campos vacíos:

python
```
if nombre == "" or poblacion == "" or superficie == "" or continente == "":
    print("Error: no se permiten campos vacíos.")
    return
```

Validación de tipos de datos:

python
```
if not poblacion.isdigit() or not superficie.isdigit():
    print("Error: población y superficie deben ser números enteros.")
    return
```

Búsqueda de países:

python
```
if termino in pais["nombre"].lower():
    encontrados.append(pais)
```

Menú de opciones:

python
```
if opcion == "1":
    agregar_pais(catalogo)
elif opcion == "2":
    actualizar_pais(catalogo)
```

 # ... más opciones

Filtrado por criterios:

python
```
if opcion == "1":  # Filtrar por continente
   # Código específico
elif opcion == "2":  # Filtrar por población
   # Código específico
```

Los condicionales permiten que el programa valide entradas, tome decisiones según la opción del usuario y maneje diferentes casos de uso de forma robusta.

Ordenamientos

Introducción

Los algoritmos de ordenamiento reorganizan los elementos de una estructura de datos según un criterio específico (ascendente o descendente). Son fundamentales para presentar información de manera organizada y facilitar búsquedas eficientes.

Funciones y utilidad en programación

- **Organización de datos:** Presentan información de forma estructurada y comprensible
- **Facilitan búsquedas:** Los datos ordenados permiten búsquedas más eficientes
- **Análisis de información:** Ayudan a identificar valores extremos (máximos/mínimos)
- **Preparación de reportes:** Ordenar datos antes de presentarlos mejora la experiencia del usuario

Métodos de ordenamiento en Python

Python proporciona métodos integrados para ordenar listas:

python
```
# Método sort() - modifica la lista original
lista.sort()                # Ascendente
lista.sort(reverse=True)    # Descendente

# Función sorted() - retorna nueva lista ordenada
nueva_lista = sorted(lista)

# Ordenamiento con key (criterio personalizado)
lista.sort(key=lambda x: x["campo"])
```

Relación con nuestro proyecto

La función `ordenar_paises()` implementa ordenamiento con múltiples criterios:

python
```
def ordenar_paises(catalogo):
   # Usuario selecciona criterio
   print("1. Por nombre")
   print("2. Por población")
   print("3. Por superficie")
```

```
# Usuario selecciona sentido
sentido = input("Orden ascendente (a) o descendente (d): ")
ascendente = True if sentido != "d" else False

# Ordenamiento según criterio
if opcion == "1":
    catalogo.sort(key=lambda x: x["nombre"], reverse=not ascendente)
elif opcion == "2":
    catalogo.sort(key=lambda x: x["poblacion"], reverse=not ascendente)
elif opcion == "3":
    catalogo.sort(key=lambda x: x["superficie"], reverse=not ascendente)
```

Características importantes:

- Usa el método `sort()` que modifica el catálogo original
- Utiliza **funciones lambda** como parámetro `key` para especificar el campo de ordenamiento
- El parámetro `reverse` controla si el orden es ascendente (False) o descendente (True)
- El ordenamiento es **estable**, manteniendo el orden relativo de elementos iguales
- Permite ordenar por nombre (alfabéticamente), población o superficie (numéricamente)

Estadísticas Básicas

Introducción

Las estadísticas básicas son cálculos matemáticos que resumen y describen características importantes de un conjunto de datos. Permiten obtener información significativa de grandes volúmenes de información mediante indicadores numéricos.

Funciones y utilidad en programación

- **Análisis de datos:** Proporcionan métricas clave sobre el conjunto de información
- **Identificación de tendencias:** Revelan patrones y valores representativos
- **Comparaciones:** Facilitan la comparación entre diferentes grupos de datos
- **Toma de decisiones:** Proveen información cuantitativa para decisiones informadas

Operaciones estadísticas comunes

python

```
# Promedio (media aritmética)
promedio = suma_valores / cantidad_elementos

# Valor máximo y mínimo
maximo = max(lista)
minimo = min(lista)

# Conteo y frecuencias
contador = {}
for elemento in lista:
```

```
    if elemento in contador:
        contador[elemento] += 1
    else:
        contador[elemento] = 1
```

Relación con nuestro proyecto

La función `mostrar_estadisticas()` calcula múltiples indicadores:

```
python
def mostrar_estadisticas(catalogo):
    # Inicialización de variables
    mayor_pob = catalogo[0]
    menor_pob = catalogo[0]
    total_pob = 0
    total_sup = 0
    continentes = {}

    # Recorrido y cálculos
    for p in catalogo:
        total_pob += p["poblacion"]
        total_sup += p["superficie"]

        # Encontrar máximo
        if p["poblacion"] > mayor_pob["poblacion"]:
            mayor_pob = p

        # Encontrar mínimo
        if p["poblacion"] < menor_pob["poblacion"]:
            menor_pob = p

        # Contar por continente
        cont = p["continente"]
        if cont in continentes:
            continentes[cont] += 1
        else:
            continentes[cont] = 1

    # Cálculo de promedios
    prom_pob = total_pob / len(catalogo)
    prom_sup = total_sup / len(catalogo)
```

Estadísticas calculadas:

- **Valor máximo:** País con mayor población
- **Valor mínimo:** País con menor población
- **Promedio de población:** Media aritmética de poblaciones
- **Promedio de superficie:** Media aritmética de superficies
- **Distribución por categoría:** Cantidad de países por continente usando un diccionario como contador

Archivos CSV

Introducción

CSV (Comma Separated Values) es un formato de archivo de texto plano que almacena datos tabulares. Cada línea representa un registro y los campos están separados por comas. Es uno de los formatos más utilizados para intercambio de datos por su simplicidad y compatibilidad universal.

Funciones y utilidad en programación

- **Intercambio de datos:** Formato estándar para transferir información entre sistemas diferentes
- **Compatibilidad:** Legible por la mayoría de programas de hojas de cálculo (Excel, Google Sheets)
- **Simplicidad:** Fácil de leer, escribir y procesar programáticamente
- **Eficiencia:** Bajo costo computacional para manejar grandes volúmenes de datos
- **Persistencia:** Permiten almacenar datos permanentemente fuera de la memoria del programa

Estructura de un archivo CSV

```
nombre,poblacion,superficie,continente
Argentina,45376763,2780400,América
Japón,125800000,377975,Asia
Brasil,213993437,8515767,América
```

Operaciones básicas con archivos

```
python
# Apertura con context manager (recomendado)
with open("datos.csv", "r") as archivo:
    contenido = archivo.read()

# Modos de apertura
"r"  # Lectura: archivo debe existir
"w"  # Escritura: crea o sobrescribe
"a"  # Agregar: añade al final sin borrar
```

Relación con nuestro proyecto

Aunque nuestro proyecto actualmente **no implementa la carga desde CSV**, la consigna requiere esta funcionalidad. El proyecto está diseñado para incorporarla fácilmente:

Función de carga implementable:

```
python
def cargar_catalogo():
    catalogo = []
    try:
        with open("paises.csv", "r", encoding="utf-8") as archivo:
            lineas = archivo.readlines()
            # Saltar encabezado
            for i in range(1, len(lineas)):
                linea = lineas[i].strip()
                if linea:  # Ignorar líneas vacías
                    campos = linea.split(",")
```

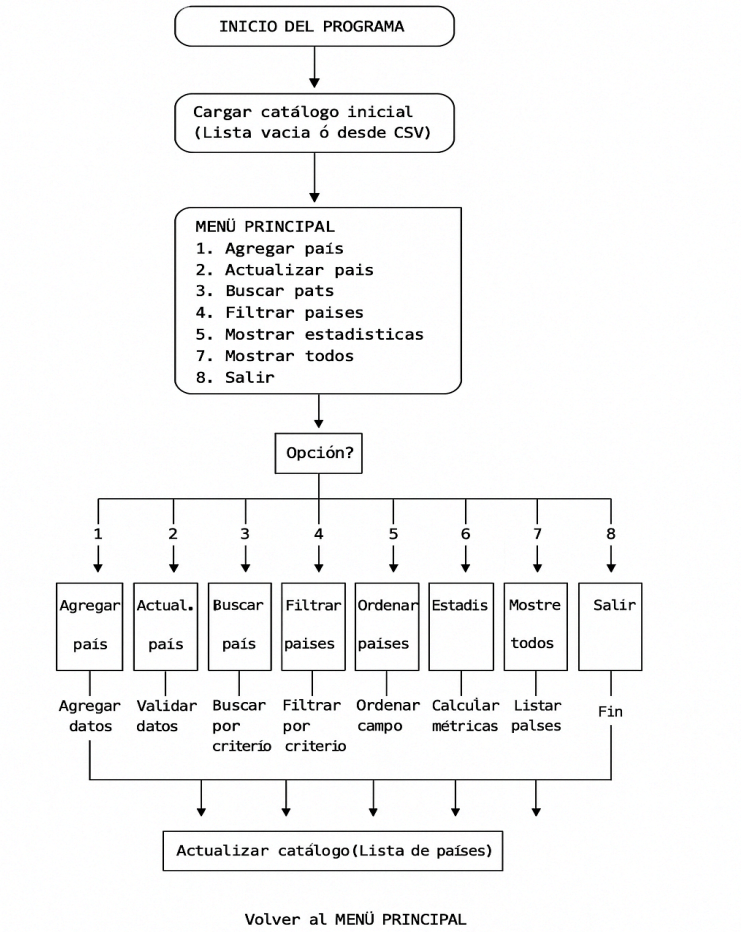
```
                pais = {
                    "nombre": campos[0],
                    "poblacion": int(campos[1]),
                    "superficie": int(campos[2]),
                    "continente": campos[3]
                }
                catalogo.append(pais)
        print(f"{len(catalogo)} países cargados correctamente.")
    except FileNotFoundError:
        print("Archivo CSV no encontrado. Se inicia catálogo vacío.")
    except Exception as e:
        print(f"Error al cargar CSV: {e}")

    return catalogo
```

Funcionalidad futura requerida:

- **Lectura:** Cargar datos iniciales desde `paises.csv`
- **Validación:** Verificar formato correcto y tipos de datos
- **Manejo de errores:** Capturar excepciones (archivo no encontrado, formato incorrecto)
- **Conversión de tipos:** Transformar strings a enteros para población y superficie

Diagrama de Flujo de Operaciones



Conclusiones

Aprendizajes sobre Estructuras de Datos

El desarrollo de este proyecto nos permitió comprender la importancia de seleccionar estructuras de datos adecuadas. La combinación de listas y diccionarios resultó fundamental:

- Las listas mantuvieron colecciones ordenadas permitiendo recorridos secuenciales del catálogo de países y almacenamiento dinámico mediante append().
- Los diccionarios representaron cada país con atributos claramente identificados, logrando acceso directo en tiempo constante O(1) y código autodocumentado.
- La sinergia lista-diccionario nos enseñó a combinar estructuras estratégicamente, aprovechando las fortalezas de cada una para modelar sistemas complejos de forma eficiente.

Modularidad y Reutilización de Código

La implementación de ocho funciones especializadas transformó nuestra forma de abordar problemas complejos:

- **Dividir la complejidad:** Cada función con una responsabilidad única facilitó el desarrollo y mantenimiento del código.
- **Facilitar el debugging:** Funciones pequeñas con ámbitos bien definidos permitieron localizar y corregir errores rápidamente.
- **Promover la reutilización:** Pasar el catálogo como parámetro eliminó duplicación y garantizó consistencia en todas las operaciones.

Este enfoque modular no solo mejoró la calidad técnica, sino que cambió nuestra mentalidad: aprendimos a pensar en componentes reutilizables con responsabilidades delimitadas.

Estadísticas y Análisis de Datos

La función mostrar_estadisticas() nos introdujo al análisis de datos programático:

- **Promedios:** Calcular población y superficie promedio nos enseñó a sintetizar conjuntos de datos en valores representativos.
- **Extremos:** Identificar máximos y mínimos mediante recorridos lineales mostró la efectividad de algoritmos simples bien aplicados.
- **Conteo por categorías:** Usar diccionarios como contadores para agrupar países por continente reveló técnicas poderosas para analizar distribuciones en datasets categóricos.

Comprendimos que las estadísticas son herramientas para extraer conocimiento significativo de grandes volúmenes de información.

Validación y Robustez

Implementar validaciones exhaustivas nos demostró que un programa funcional debe ser robusto ante entradas incorrectas:

- Validar campos vacíos previno errores de integridad.
- Verificar tipos de datos evitó excepciones durante operaciones numéricas.
- Manejar búsquedas sin resultados mejoró la experiencia del usuario.

Esta práctica nos enseñó que la programación defensiva distingue código amateur de código profesional.

Ordenamiento y Organización
La implementación de múltiples criterios de ordenamiento nos familiarizó con conceptos algorítmicos importantes:

- Comprender la diferencia entre sort() (modifica original) y sorted() (retorna copia) nos enseñó a considerar efectos secundarios.
- Los parámetros key y reverse demostraron la flexibilidad de APIs bien diseñadas.

Trabajo Colaborativo
Más allá de lo técnico, el proyecto nos preparó para el desarrollo profesional:

- **Comunicación:** Discutir decisiones de diseño y distribuir tareas fortaleció habilidades interpersonales.
- **Revisión de código:** Detectar errores y sugerir mejoras mantuvo estándares de calidad.
- **Documentación:** Aprendimos que documentar es parte integral del desarrollo, no un añadido opcional.

Reflexión Final
Este trabajo integrador marcó un punto de inflexión en nuestra formación. Pasamos de resolver ejercicios aislados a construir un sistema completo y funcional que integra múltiples conceptos coherentemente.

Comprendimos que la programación requiere tanto rigor técnico como creatividad para diseñar soluciones elegantes y mantenibles. El proyecto nos enfrentó a decisiones reales de diseño que determinan la diferencia entre código frágil y robusto.

La experiencia colaborativa nos enseñó que el desarrollo de software es fundamentalmente social: las habilidades técnicas son necesarias pero no suficientes. La comunicación y el trabajo en equipo son igualmente esenciales para el éxito profesional.

Nos sentimos preparados para afrontar desafíos más complejos, llevando conocimientos técnicos y una mentalidad profesional orientada a la calidad, claridad y colaboración.

Fuentes Bibliográficas

1. **Material didáctico de la cátedra:**
    - Material PDF "Listas - Unidad 5", Cátedra Programación 1, UTN
    - Material PDF "Datos Complejos - Unidad 7", Cátedra Programación 1, UTN

    - Material PDF "Funciones - Unidad 6", Cátedra Programación 1, UTN
    - Notebook "Introducción a las listas de Python", Cátedra Programación 1, UTN
    - Notebook "Diccionarios", Cátedra Programación 1, UTN
    - Notebook "Funciones", Cátedra Programación 1, UTN

2. **Python Software Foundation.** (2024). *Python 3.12 Documentation - Data Structures*. Disponible en: https://docs.python.org/3/tutorial/datastructures.html

3. **Apinem.** (2024). *Listas en programación: qué son y para qué sirven*. Disponible en: https://www.apinem.com/listas-programacion-que-son-y-para-que-sirven/
    - Guía práctica sobre listas en programación con ejemplos aplicados.

4. **Apinem.** (2024). *Funciones en programación: 3 pasos para declarar una función*. Disponible en: https://www.apinem.com/funciones-en-programacion/#3-pasos-para-declarar-una-funcion
    - Tutorial sobre declaración y uso de funciones en programación.

5. **Geeknetic.** (2024). *Archivo CSV: qué es y para qué sirve*. Disponible en: https://www.geeknetic.es/Archivo-CSV/que-es-y-para-que-sirve
    - Explicación detallada del formato CSV y sus aplicaciones prácticas.