sustantiva
CON•SENTIDO

Estructuras de Datos en Python

Introducción

Introducción

- Las estructuras de datos son componentes esenciales en el diseño algorítmico.
- Elegir la estructura adecuada mejora la eficiencia, legibilidad y mantenimiento del código.
- Python proporciona estructuras integradas que cubren necesidades textuales, numéricas, asociativas y matemáticas.

Cadenas de Texto (Strings)

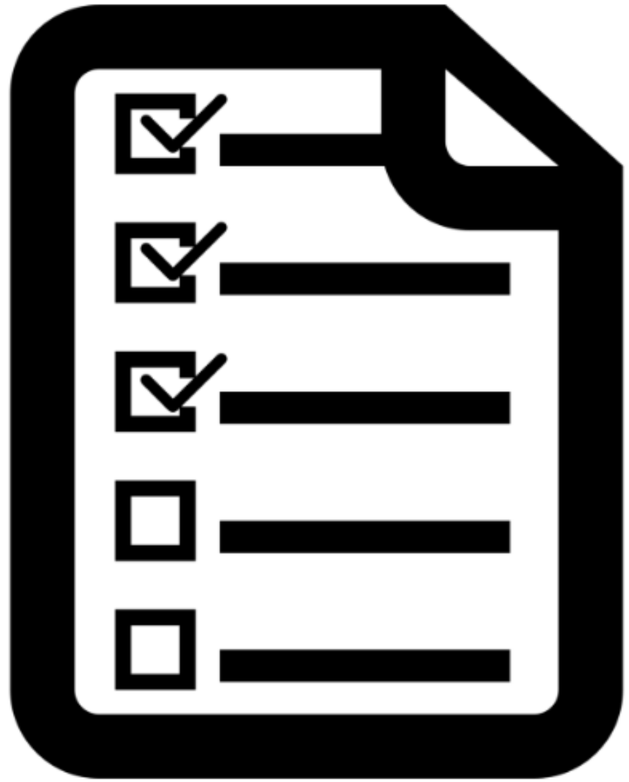
Cadenas de Texto (Strings)

- Estructuras **inmutables** utilizadas para representar y manipular texto.
- Permiten operaciones como búsqueda, reemplazo, transformación y separación.
- Útiles en entrada de datos, validación, análisis léxico y procesamiento textual.
- Métodos comunes: .upper(), .replace(), .split(), expresiones regulares.

Listas (list)

Listas (list)

- Estructuras **mutables y ordenadas** para almacenar elementos heterogéneos.
- Se adaptan bien a colecciones dinámicas, buffers o estructuras tipo pila y cola.
- Soportan inserciones, eliminaciones, ordenamientos y recorridos personalizados.
- Métodos clave: .append(), .insert(), .remove(), .sort(), .reverse().



Tuplas (tuple)

Tuplas (tuple)

- Similares a las listas, pero inmutables.
- Representan datos que no deben modificarse: coordenadas, fechas, constantes.
- Garantizan estabilidad referencial y acceso más rápido.
- Son óptimas para retornos múltiples o estructuras protegidas en memoria.

Diccionarios (dict)

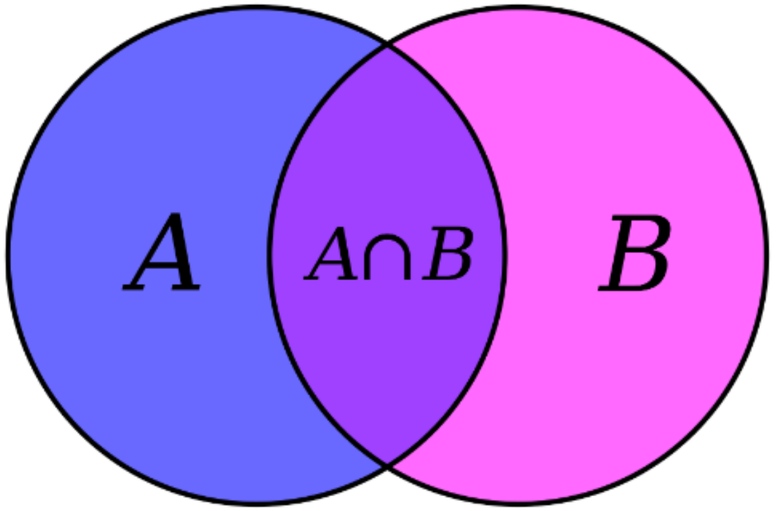
Diccionarios (dict)

- Colecciones **asociativas** de pares clave–valor, mutables y no ordenadas (hasta Python 3.6).
- Ideales para estructuras tipo JSON, bases de datos pequeñas, configuraciones.
- Acceso muy eficiente mediante hashing (O(1)).
- Métodos útiles: .get(), .items(), .keys(), .values().

Conjuntos (sets)

Conjuntos (sets)

- Estructuras **no ordenadas y sin duplicados**, mutables.
- Ideales para modelar conjuntos matemáticos: unión, intersección, diferencia.
- También útiles para eliminar duplicados o validar membresías rápidamente.
- Operaciones: .union(), .intersection(), .difference(), in.



Criterios para Elegir la Estructura Adecuada

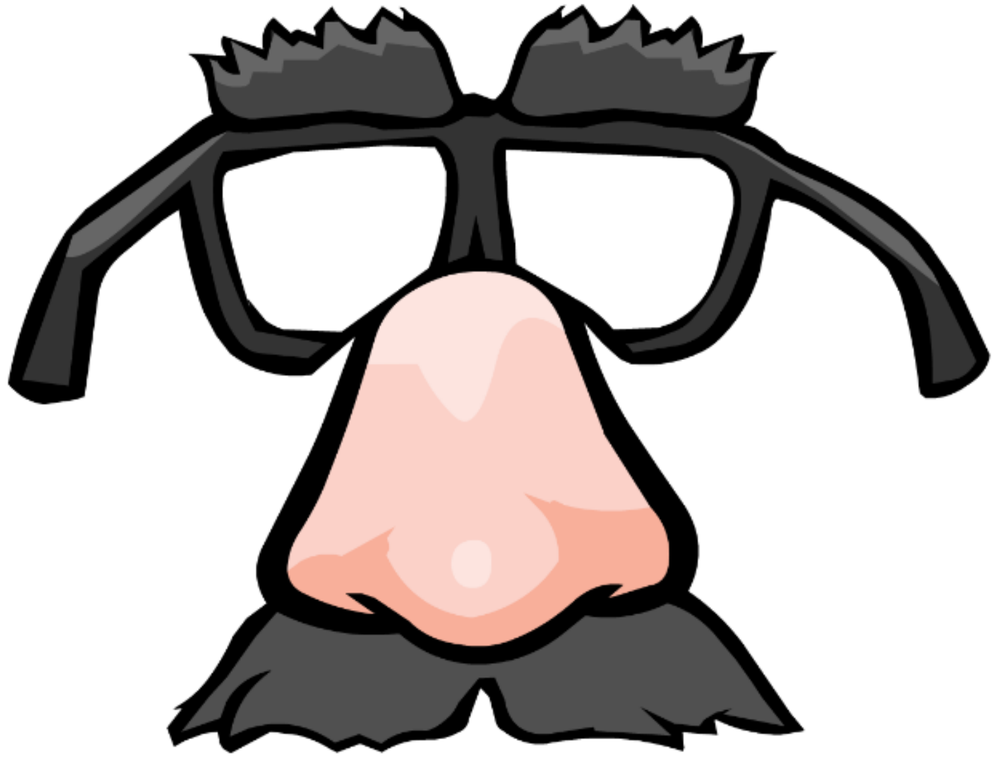
Criterios para Elegir la Estructura Adecuada

- ¿Se requiere modificación del contenido? → listas/dict/sets.
- ¿Se necesita orden y protección contra cambios? → tuplas.
- ¿Se accede por posición o por clave?
- ¿Se necesita evitar duplicados o trabajar con colecciones únicas? → sets.
- Evaluar siempre la **complejidad computacional esperada** (O(1) vs O(n)).

Caso de Uso Integrado

Caso de Uso Integrado

- Problema: contar la frecuencia de palabras únicas en un texto.
- Uso combinado de:
  - **String** para manipulación textual.
  - **Lista** para almacenar y recorrer palabras.
  - **Diccionario** para contar frecuencias.
  - **Set** para obtener palabras únicas.
- Refleja cómo elegir estructuras mejora la claridad y eficiencia del algoritmo.



Comparación Formal de Estructuras

- **str (cadena de texto):**
  - Orden: Sí
  - Mutable: No
  - Acceso: Por índice
  - Permite duplicados: Sí
  - Uso principal: Procesamiento de texto
- **list (lista):**
  - Orden: Sí
  - Mutable: Sí
  - Acceso: Por índice
  - Permite duplicados: Sí
  - Uso principal: Manejo de datos dinámicos y colecciones modificables
- **tuple (tupla):**
  - Orden: Sí
  - Mutable: No
  - Acceso: Por índice
  - Permite duplicados: Sí
  - Uso principal: Almacenamiento de datos constantes o inmutables
- **dict (diccionario):**
  - Orden: No (en versiones recientes de Python 3.7+ sí mantiene orden de inserción, pero no es su propiedad principal)
  - Mutable: Sí
  - Acceso: Por clave
  - Permite duplicados: No en las claves
  - Uso principal: Acceso rápido a datos mediante claves únicas
- **set (conjunto):**
  - Orden: No
  - Mutable: Sí
  - Acceso: Por membresía (pertenencia)
  - Permite duplicados: No
  - Uso principal: Gestión de elementos únicos, eliminación de duplicados y operaciones de conjuntos

Conclusión

Conclusión

- La estructura de datos adecuada mejora el diseño y rendimiento del algoritmo.
- Python ofrece una diversidad de estructuras para distintos propósitos.
- Es clave practicar su uso y comprender sus ventajas computacionales.
- Combinar estructuras correctamente permite construir soluciones profesionales, limpias y escalables.


Energiza!