Conferencia 07

Diccionarios y Conjuntos

Repaso: Estructuras Secuenciales

- Listas y Tuplas.
- Acceso por **índice** (posición): `[0]`, `[1]`, `[2]...`
- El **orden** es fundamental.

Hoy: Nuevos Desafíos

1. ¿Cómo almacenar datos por “etiqueta” o “nombre” en lugar de índice?

- (Ej: `alumno['nombre']` en lugar de `alumno[0]`)

2. ¿Cómo manejar colecciones donde **no** queremos duplicados?

Las Soluciones

- **Diccionarios** (`dict`): Para el problema 1 (etiquetas).
- **Conjuntos** (`set`): Para el problema 2 (elementos únicos).
- Estas estructuras no se basan en el *orden*, sino en la *pertenencia* y la *unicidad*.

Parte 1: Diccionarios (`dict`)

- Colección de pares `clave: valor`.
- Implementación de “Mapas” o “Tablas Hash”.
- `{'nombre': 'Ana', 'id': 12345, 'activa': True}`

Diccionarios: Reglas de Claves

- Las claves deben ser **únicas**.
- Las claves deben ser **inmutables**.
- **SÍ pueden ser claves:** `str`, `int`, `float`, `bool`, `tuple`.
- **NO pueden ser claves:** `list`, `dict`, `set`.

Ejemplo de código

Operaciones: Lectura (Acceso)

- **Acceso directo:** `mi_dict['clave']`
- Rápido, pero… `KeyError` si no existe
- **Acceso seguro:** `mi_dict.get('clave', valor_por_defecto)`
- Devuelve `None` o el valor por defecto si no existe.
- No genera error.

Operaciones: Escritura (C/U)

- **Añadir / Actualizar:** `mi_dict['clave_nueva'] = 'valor'`
- Si la clave existe, actualiza el valor.
- Si no existe, crea el nuevo par.
- **Fusionar:** `mi_dict.update(otro_dict)`
- Añade/sobrescribe múltiples pares desde otro diccionario.

Operaciones: Borrar y Comprobar

- **Comprobar clave:** `'clave' in mi_dict`
- (Extremadamente rápido)
- **Eliminar (directo):** `del mi_dict['clave']`
- `KeyError` si no existe
- **Eliminar (seguro):** `valor = mi_dict.pop('clave', default)`
- Elimina la clave, devuelve el valor.
- Permite un valor por defecto si no existe.

Ejemplo de código

Iteración de Diccionarios

Hay 3 formas de recorrerlos:

- `for clave in mi_dict:`
- Itera sobre las **claves** (por defecto).
- `for valor in mi_dict.values():`
- Itera sobre los **valores**.
- `for clave, valor in mi_dict.items():`
- La más común. Itera sobre pares `(clave, valor)`.

Ejemplo de código

Parte 2: Conjuntos (`set`)

- Colección **sin orden**.
- Elementos **únicos** (no permite duplicados).

Conjuntos: Usos Principales

1. Eliminar duplicados de una lista (muy rápido).
2. Comprobar pertenencia (`in`) (muy rápido).
3. Realizar operaciones matemáticas (álgebra de conjuntos).

Creación de Conjuntos

- Con llaves: `numeros = {1, 2, 3, 2, 1}` -> `{1, 2, 3}`
- Desde un iterable: `unicos = set([1, 1, 2, 3, 1])` -> `{1, 2, 3}`

  Importante - Conjunto vacío: `mi_set = set()` - (`{}` es un diccionario vacío)

Operaciones Básicas (Set)

- **Añadir:** `mi_set.add('elemento')`
  - (No hace nada si ya existe).
- **Eliminar (directo):** `mi_set.remove('elemento')`
  - `KeyError` si no existe
- **Eliminar (seguro):** `mi_set.discard('elemento')`
  - No da error si no existe.
- **Comprobar:** `'elemento' in mi_set`

Ejemplo de código

- Crear conjuntos.
- Eliminar duplicados de una lista.
- Operaciones `.add()`, `.remove()`, `.discard()`.

Álgebra de Conjuntos

- **Unión (`|`):** `set_A | set_B` (Todos los elementos).
- **Intersección (`&`):** `set_A & set_B` (Solo los comunes).
- **Diferencia (`-`):** `set_A - set_B` (En A, pero no en B).
- **Dif. Simétrica (`^`):** `set_A ^ set_B` (En A o B, pero no en ambos).

Métodos vs. Operadores

- **Operadores (`|`, `&`, ...):**
- Requieren que *ambos* lados sean `set`.
- **Métodos (`.union()`, `.intersection()`, ...):**
- Más flexibles.
- Pueden operar con *cualquier iterable* (ej. `set.union(lista)`).

Subconjuntos y `frozenset`

- **Comprobar:**
- `set_A.issubset(set_B)` (o `set_A <= set_B`)
- `set_B.issuperset(set_A)` (o `set_B >= set_A`)
- **`frozenset`**:
- Versión **inmutable** de un conjunto.
- Al ser inmutable, SÍ puede ser usado como clave de diccionario

Ejemplo de código

Parte 3: Empaquetado de Argumentos

Conexión: Usamos `tuple` y `dict` para crear funciones más flexibles.

- `*args`
- `**kwargs`

`*args` (Argumentos Posicionales)

- `def mi_funcion(param_fijo, *args):`
- Agrupa argumentos posicionales “extra” en una **TUPLA**.
- `mi_funcion('a', 1, 2, 3)` -> `args` será `(1, 2, 3)`

`**kwargs` (Keyword Arguments)

- `def mi_funcion(**kwargs):`
- Agrupa argumentos nombrados “extra” en un **DICCIONARIO**.
- `mi_funcion(id=1, modo='dev')` -> `kwargs` será `{'id': 1, 'modo': 'dev'}`

Orden y Keyword-Only

- Orden estricto: `(std, *args, *, kw_only, **kwargs)`
- El `*` solo, fuerza a que los siguientes argumentos sean *keyword-only*.

  `def crear_usuario(nombre, *, permisos):` - `permisos` DEBE pasarse por nombre (`permisos='admin'`).

Ejemplo de código

Parte 4: Desempaquetado (* y **)

La operación inversa: usar * y ** al *llamar* funciones o *crear* colecciones.

Desempaquetado en Llamadas

- `*` (Listas/Tuplas):
- `numeros = [10, 20, 30]`
- `suma(*numeros)` equivale a `suma(10, 20, 30)`
- `**` (Diccionarios):
- `datos = {'nombre': 'Ana', 'id': 1}`
- `crear_usuario(**datos)` equivale a `crear_usuario(nombre='Ana', id=1)`

Desempaquetado para Fusión

- **Fusión de Listas:**
- `lista_total = [*lista_A, 99, *lista_B]`
- **Fusión de Diccionarios:** (Muy útil)
- `config = {**config_default, **config_usuario}`
- Si hay claves repetidas, el **último valor "gana"**.

Ejemplo de código

Resumen

- **`dict`**: `clave:valor`. Acceso seguro con `.get()`. Iterar con `.items()`.
- **`set`**: Únicos y sin orden. Para eliminar duplicados y álgebra (`|`, `&`).
- **`frozenset`**: `set` inmutable (usable como clave).
- **`*args`** / **`**kwargs`**: (En *definición*) Empaqueta argumentos en `tuple` / `dict`.
- `*` / `**`: (En *llamada* o *literal*) Desempaqueta iterables / diccionarios.

Conferencia 07

Diccionarios y Conjuntos