📝 Apuntes de Programación: Los Conjuntos (Sets) en Python

Los Conjuntos (`set`) son colecciones de datos fundamentales en Python, caracterizadas por dos reglas clave: son **desordenados** y solo almacenan **elementos únicos**. Son ideales para eliminar duplicados y realizar operaciones lógicas.

1. Creación y Características Básicas

A. Creación y Unicidad

Los conjuntos se pueden crear de dos maneras, pero su característica principal es que eliminan automáticamente los duplicados.

Código de Ejemplo	Descripción y Resultado
`conjunto1 = {"Ana","Pedro","Luis","Eva","Ana"}`	**Uso de llaves** `{}` **:** Es la forma más común. Aunque "Ana" se repite, solo se almacena una vez.
`conjunto2 = set(["Ana","Pedro","Luis","Eva","Ana"])`	**Uso del constructor** `set()` **:** Convierte otra secuencia (como una lista) en un conjunto, eliminando los duplicados en el proceso.

B. Elementos y Tamaño

- **Desordenados:** Los conjuntos no mantienen el orden de inserción. Cada vez que se imprime el conjunto, el orden puede variar.
- **Longitud:** La función `len()` devuelve el número de elementos únicos.

```
print(len(conjunto1))
```

2. Operaciones Fundamentales

A. Agregar y Eliminar Elementos

Método	Propósito	Comportamiento Clave
`conjunto.add("elemento")`	Añadir un solo elemento al conjunto.	Si el elemento ya existe, no hace nada (no añade duplicados).
`conjunto.remove("elemento")`	Elimina un elemento.	**¡Lanza un error (** `KeyError` **) si el elemento no existe!**
`conjunto.discard("elemento")`	Elimina un elemento.	**No lanza error** si el elemento no existe. Es más seguro que `remove()` .
`conjunto.pop()`	Elimina y devuelve un elemento **aleatorio** del conjunto (debido a	

	que no tienen orden).
`conjunto.clear()`	Elimina todos los elementos, dejando el conjunto vacío.

B. Iteración y Verificación

- **Recorrer:** Se pueden recorrer con un bucle `for` normal. El orden será aleatorio.

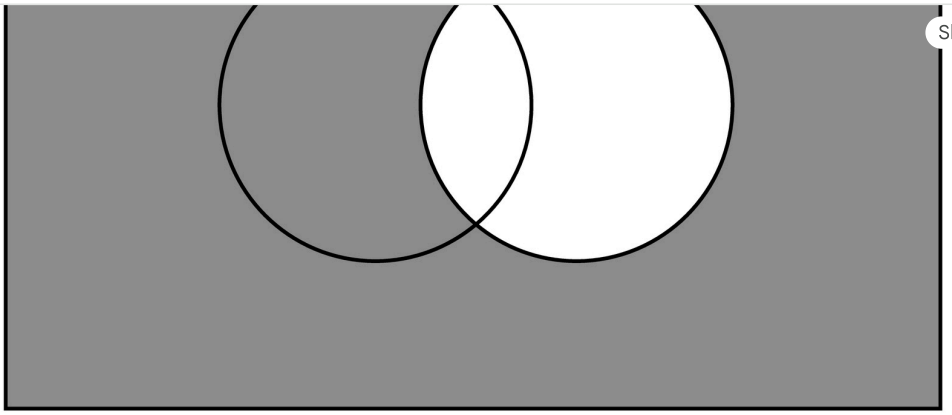
```
for nombre in conjunto1:
    print(nombre)
```

- **Verificar Existencia:** El operador `in` es extremadamente rápido para verificar si un elemento existe.

```
if "Ana" in conjunto1:
    print("Ana esta")
```

3. Operaciones Matemáticas con Conjuntos (Álgebra de Conjuntos)

La principal fortaleza de los conjuntos es su capacidad para realizar operaciones lógicas de manera nativa, usando tanto operadores especiales como métodos.



Operación	Operador	Método Equivalente	Descripción
Intersección	`&`	`set1.intersection(set2)`	Devuelve un nuevo conjunto con los elementos **comunes** a ambos conjuntos.
Unión	`	`	`set1.union(set2)`
Diferencia	`-`	`set1.difference(set2)`	Devuelve los elementos que están en `set1` **pero no** en `set2` .

Ejemplos del Código (Profesores)

```
profesPrimero = {"Natalia","José María","Pedro","Yago"}
profesSegundo =  {"José María","Agustín","Puche","Pedro"}
```

1. **Intersección (`&`):**

```
print(profesPrimero & profesSegundo) # Resultado: {'Pedro', 'José María'}
```

2. **Unión (`|`):**

```
print(profesPrimero | profesSegundo) # Resultado: {'Natalia', 'Yago', 'José María',
```

3. **Diferencia (`-`):**

```
print(profesSegundo - profesPrimero) # Resultado: {'Agustín', 'Puche'} (Los que est
```

4. Consejos para Mejorar la Programación con Conjuntos (Examen Pro-Tip)

🎯 Principio 1: Eliminar Duplicados de Forma Eficiente

Si tu objetivo es simplemente obtener una lista sin duplicados, convertir la lista a un conjunto y luego de vuelta a una lista es la forma más rápida y legible de hacerlo.

- **Ejemplo:** `lista_sin_duplicados = list(set(mi_lista))`

🎯 Principio 2: La Verificación de Pertenencia es Casi Instantánea

Cuando tienes que revisar repetidamente si un elemento existe en una colección grande (millones de elementos), usar un **conjunto** (`set`) es mucho más rápido que usar una **lista** (`list`).

- **Razón:** Los conjuntos están optimizados para esta operación, lo que se conoce como complejidad de tiempo O(1) (constante). En contraste, una lista tiene que buscar elemento por elemento (O(n)).

🎯 Principio 3: Usar `discard` sobre `remove`

A menos que la no existencia de un elemento sea un error crítico en tu lógica, usa siempre `conjunto.discard()` para eliminar elementos. Si el elemento no existe, `discard` simplemente ignora la operación y evita que tu programa falle, a diferencia de `remove` .

🎯 Principio 4: Conjuntos Inmutables (`frozenset`)

Si necesitas una colección de elementos únicos que no pueda ser modificada después de su creación (inmutable), puedes usar `frozenset()` . Lo necesitas, por ejemplo, si quieres usar un conjunto como la clave de un diccionario (ya que las claves deben ser inmutables).