Python: conjuntos y diccionarios

Sistemas de gestión empresarial – 148fa (DAM)

Tipos de datos complejos

- **Tuplas**: variables que guardan datos que no pueden ser modificados. Los datos pueden ser de diferentes tipos .

- **Listas**: son variables que guardan datos de diferentes tipos, pero que sí que pueden ser modificados.

- **Diccionarios**: colección de pares clave-valor. Cada clave es única y se usa para acceder a su valor asociado. Los diccionarios son mutables, lo que significa que sus valores pueden cambiarse después de su creación.

- **Conjuntos**: colección desordenada de elementos únicos, es decir, no permite duplicados. Los conjuntos son mutables, pero sus elementos deben ser inmutables (como números, cadenas, o tuplas).

Diccionarios - Características

- En listas y tuplas se accede a la información mediante los índices numéricos. En el diccionario se accede a la información a través de las claves asociadas.
- Las **claves son únicas**: no puede haber claves repetidas. Los valores sí se pueden repetir.
- No hay forma de acceder a una clave a través de su valor.
- Las **claves** pueden ser: cadenas, enteros, tuplas… variables de tipo **inmutable.**
- Los **valores** pueden ser: listas, cadenas, otros diccionarios…

Diccionarios - Características

```
mi_diccionario= {'a': 1, 'b': 2, 'c': 3}


clases = {}
clases["lunes"] = ['SGE', 'SOM']
clases["martes"] = ['ED']
clases["miércoles"] = ['SGE', 'MME']
clases["jueves"] = []
clases["viernes"] = ['DI']
print(clases['miércoles'])
```

Diccionarios - Características

```
estudiante = {"nombre": "Iñaki Perurena",
              "edad": 30,
              "nota_media": 7.25,
              "repetidor" : False
              }
# Acceder al valor de una clave
dad = estudiante["edad"] # devuelve el valor de 'edad'
# nota_media = estudiante.get("nota_media") # devuelve el valor de 'nota_media'
# Insertar o actualizar un valor:
estudiante["edad"] = 25 # actualiza el valor de 'edad'
estudiante["suspensos"] = 3 # inserta una nueva pareja clave - valor
# insertar una pareja clave - valor o actualizar si ya existe:
estudiante.update({'aprobados':'8'})
```

Diccionarios - Métodos

- **diccionario.keys()**: Devuelve todas las claves del diccionario.

```
mi_diccionario =  {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
claves= mi_diccionario.keys()

claves → ['a','b','c','d']
```

- **diccionario.values()**: Devuelve todos los valores del diccionario.

```
mi_diccionario =  {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}
valores= mi_diccionario.values()

valores→ [1,2,3,4]
```

Diccionarios - Métodos

- **diccionario.pop(clave[,<default>]):** Elimina la clave del diccionario y devuelve su valor asociado. Si no la encuentra y se indica un valor por defecto, devuelve el valor por defecto indicado.

```
mi_diccionario =  {'a' : 1, 'b' : 2, 'c' : 3 , 'd' : 4}

valor = mi_diccionario.pop('b')



valor → 2

mi_diccionario → {'a' : 1, 'c' : 3 , 'd' : 4}
```

Diccionarios - Métodos

- **diccionario.get(clave):** Recibe como parámetro una clave, devuelve el valor de la clave. Si no lo encuentra, devuelve un objeto none.

```
mi_diccionario =  {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}

valor = mi_diccionario.get(‘b’)


valor → 2
```

- **diccionario.clear():** Vacía el diccionario.

```
mi_diccionario =  {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}

mi_diccionario.clear()


mi_diccionario → { }
```

Diccionarios - Métodos

- **zip():** Recibe como parámetro dos elementos iterables (cadena o lista). Ambos parámetros deben tener el mismo número de elementos. Se devolverá un diccionario relacionando el elemento i-ésimo de cada uno de los iterables.

```
mi_diccionario = dict(zip('abcd',[1,2,3,4]))


mi_diccionario → {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
```

- **diccionario.copy():** Hace una copia del diccionario original. Se puede tratar de manera independiente.

```
dic = {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}

dic1 = dic.copy()


dic1 → {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}
```

Diccionarios - Métodos

- **diccionario.update(diccionario2):** Recibe como parámetro otro diccionario. Si se tienen claves iguales, actualiza el valor de la clave repetida; si no hay claves iguales, se inserta este par clave-valor al diccionario.

```
dic1 = {‘a’ : 1, ’b’ : 2, ‘c’ : 3 , ‘d’ : 4}

dic2 = {‘c’ : 6, ’b’ : 5, ‘e’ : 9 , ‘f’ : 10}

dic1.update(dic2)



dic1 → {‘a’ : 1, ’b’ : 5, ‘c’ : 6 , ‘d’ : 4 , ‘e’ : 9 , ‘f’ : 10}
```

Diccionarios - Métodos

- **clave in diccionario:** Devuelve True si el diccionario contiene la clave o False en caso contrario.
- **valor in diccionario.values():** Devuelve True si el diccionario contiene el valor o False en caso contrario.

```
mi_diccionario = {
    "nombre": "Ana",
    "edad": 25,
    "ciudad": "Madrid"
}
# 1. clave in diccionario
# Verifica si una clave está en el diccionario
print("nombre" in mi_diccionario)  # Resultado: True (la clave "nombre" existe)
print("apellido" in mi_diccionario)  # Resultado: False (la clave "apellido" no existe)

# 2. valor in diccionario.values()
# Verifica si un valor está en el diccionario
print("Ana" in mi_diccionario.values())  # Resultado: True (el valor "Ana" existe)
print(30 in mi_diccionario.values())  # Resultado: False (el valor 30 no existe)
```

Diccionarios - Cómo recorrerlo

- **Por clave:**

```
# Diccionario de ejemplo
mi_diccionario = {
    "nombre": "Ana",
    "edad": 25,
    "ciudad": "Madrid"
}
# Recorrer las claves
for clave in mi_diccionario:
    print(clave)

for clave in mi_diccionario.keys():
        print(clave)
```

'

- **Por valor:**

```
# Diccionario de ejemplo
mi_diccionario = {
    "nombre": "Ana",
    "edad": 25,
    "ciudad": "Madrid"
}
# Recorrer los valores
for valor in mi_diccionario.values():
    print(valor)
```

Diccionarios - Cómo recorrerlo

- **Por clave y valor:**
  - **mi_diccionario.items() devuelve un iterados que produce tuplas con la clave y el valor**

```python
# Diccionario de ejemplo
mi_diccionario = {
    "nombre": "Ana",
    "edad": 25,
    "ciudad": "Madrid"
}
# Recorrer claves y valores
for clave, valor in mi_diccionario.items():
    print(f"Clave: {clave}, Valor: {valor}")
```

Diccionarios - Cómo recorrerlo

- **Por clave y valor e índice:**
  - enumerate() toma un iterable y lo recorre, devolviendo un contador (el índice) y el valor de cada elemento iterable (que en este caso es una tupla).

```python
# Diccionario de ejemplo
mi_diccionario = {
    "nombre": "Ana",
    "edad": 25,
    "ciudad": "Madrid"
}
# Recorrer claves y valores con índice
for indice, (clave, valor) in enumerate(mi_diccionario.items()):
    print(f"Índice: {indice}, Clave: {clave}, Valor: {valor}")
```

Conjuntos - Características

- Un **conjunto** (set) es una colección desordenada y **sin duplicados** de elementos. Se usa para almacenar elementos únicos y realizar operaciones matemáticas como la unión, intersección y diferencia entre conjuntos.
  - **Elementos únicos**: Los conjuntos no permiten elementos duplicados. Si intentas agregar un elemento repetido, se ignorará.
  - **Desordenados**: No mantienen un orden específico. El orden de los elementos no es garantizado.
  - **Mutables**: Puedes agregar y eliminar elementos de un conjunto.
  - **No indexados**: No puedes acceder a los elementos a través de índices como en las listas o tuplas.

Conjuntos - Métodos

- **add(elemento):** Añade un solo elemento al conjunto. Si el elemento ya existe, no lo agrega.

```
mi_conjunto = {1, 2, 3, 4, 5}
mi_conjunto.add(3)
```

- **remove(elemento):** Elimina un elemento del conjunto. Si el elemento no existe, lanza un error KeyError.

```
mi_conjunto.remove(3)
```

- **discard(elemento):** Elimina un elemento del conjunto si existe. Si no existe, no hace nada (no lanza error).

```
mi_conjunto.discard(3)
```

- **clear():** Elimina todos los elementos del conjunto.

```
mi_conjunto.clear()
```

Conjuntos - Métodos

- **union(otro_conjunto):** Devuelve un nuevo conjunto que es la unión de dos conjuntos.
- **intersection(otro_conjunto):** Devuelve un nuevo conjunto con los elementos comunes entre dos conjuntos.
- **difference(otro_conjunto):** Devuelve un nuevo conjunto con los elementos que están en el primer conjunto pero no en el segundo.

```
conjunto_1 = {1, 2, 3, 4, 5}
conjunto_2 = {3, 4, 5}
conjunto_1.union(conjunto_2)
conjunto_1.intersection(conjunto_2)
conjunto_1.difference(conjunto_2)
```

Conjuntos - Métodos

- **issubset(otro_conjunto):** Devuelve True si el conjunto actual es un subconjunto del otro.
- **issuperset(otro_conjunto):** Devuelve True si el conjunto actual es un superconjunto del otro.

```
conjunto_1 = {1, 2, 3, 4, 5}
conjunto_2 = {3, 4, 5}
conjunto_1.issubset(conjunto_2)
conjunto_1.issuperset(conjunto_2)
```

Conjuntos - Cómo recorrerlo

```python
mi_conjunto = {1, 2, 3, 4, 5}
# Recorrer un conjunto
for elemento in mi_conjunto:
    print(elemento)
```