Módulo 5.2: Iterar diccionarios

Prof. Carlos Cedeño

Objetivos de la Clase

- Comprender y utilizar los diferentes métodos para iterar sobre un diccionario.
- Aprender a buscar claves (`keys`) y valores (`values`) de manera eficiente.
- Aplicar estos conocimientos para resolver problemas prácticos.

1. ¿Por Qué Iterar en un Diccionario?

A menudo, no solo queremos acceder a un único valor a través de su clave, sino que necesitamos examinar todos los pares clave-valor de un diccionario para:

- Buscar un dato específico.
- Filtrar elementos que cumplan una condición.
- Realizar cálculos con los valores.
- Crear un nuevo diccionario a partir de uno existente.

2. Formas de Iterar en un Diccionario

Veamos las tres formas principales de recorrer un diccionario usando un bucle `for`.

a) Iterar sobre las Claves (Keys)

Esta es la forma **predeterminada** de iteración. Si usas un `for` directamente sobre el diccionario, recorrerás sus claves.

```
inventario = {'manzanas': 50, 'naranjas': 35, 'bananas': 40}

print("Iterando sobre las claves:")
for fruta in inventario:
    print(f"Clave: {fruta}")
    # Para acceder al valor, usamos la clave
    print(f"Valor: {inventario[fruta]}")
    print("---")
```

También puedes ser más explícito usando el método `.keys()`, que devuelve un objeto vista con todas las claves.

```
print("\nUsando el método .keys():")
for fruta in inventario.keys():
    print(f"Clave: {fruta}")
```

Resultado de ambos:

```
Clave: manzanas
Clave: naranjas
Clave: bananas
```

b) Iterar sobre los Valores (Values)

Si solo te interesan los valores y no necesitas las claves, puedes usar el método `.values()`.

```
inventario = {'manzanas': 50, 'naranjas': 35, 'bananas': 40}

print("Iterando sobre los valores:")
for cantidad in inventario.values():
    print(f"Cantidad en stock: {cantidad}")
```

Resultado:

```
Cantidad en stock: 50
Cantidad en stock: 35
Cantidad in stock: 40
```

c) Iterar sobre Pares Clave-Valor (Items)

Esta es la forma **más común y eficiente** cuando necesitas tanto la clave como el valor al mismo tiempo. El método `.items()` devuelve un objeto vista que contiene tuplas `(clave, valor)`.

```
inventario = {'manzanas': 50, 'naranjas': 35, 'bananas': 40}

print("Iterando sobre clave y valor con .items():")
for fruta, cantidad in inventario.items():
    print(f"Hay {cantidad} unidades de {fruta}")
```

Resultado:

```
Hay 50 unidades de manzanas
Hay 35 unidades de naranjas
Hay 40 unidades de bananas
```

Nota: El desempaquetado de tuplas (`for fruta, cantidad in ...`) hace que el código sea muy legible y eficiente.

3. Búsqueda de Elementos en un Diccionario

Ahora que sabemos cómo iterar, veamos las formas de buscar elementos.

a) Búsqueda por Clave (La forma más rápida)

La principal ventaja de los diccionarios es la búsqueda casi instantánea por clave. Para verificar si una clave existe, usamos el operador `in`.

```python
contactos = {'Ana': '555-1234', 'Juan': '555-5678', 'Pedro': '555-8765'}

# Verificar si una clave existe
if 'Ana' in contactos:
    print(f"El teléfono de Ana es {contactos['Ana']}")
else:
    print("Ana no está en la lista de contactos.")

if 'Carlos' in contactos:
    print("Carlos está en la lista.")
else:
    print("Carlos no está en la lista de contactos.")
```

Resultado:

```
El teléfono de Ana es 555-1234
Carlos no está en la lista de contactos.
```

¡Cuidado! Intentar acceder a una clave que no existe (`contactos['Carlos']`) producirá un error `KeyError` .

b) Acceso Seguro con el Método `.get()`

Para evitar el `KeyError`, podemos usar el método `.get()`. Este método devuelve el valor si la clave existe, y `None` (o un valor por defecto que especifiquemos) si no existe.

```
contactos = {'Ana': '555-1234', 'Juan': '555-5678'}

# Búsqueda segura
telefono_pedro = contactos.get('Pedro')
print(f"Teléfono de Pedro: {telefono_pedro}")

# Búsqueda segura con un valor por defecto
telefono_sofia = contactos.get('Sofia', 'Contacto no encontrado')
print(f"Teléfono de Sofia: {telefono_sofia}")
```

Resultado:

```
Teléfono de Pedro: None
Teléfono de Sofia: Contacto no encontrado
```

c) Búsqueda por Valor

No hay un método directo para buscar por valor, ya que los valores pueden estar duplicados. La única forma es **iterar sobre el diccionario** y comprobar el valor en cada paso.

```
calificaciones = {'Matemáticas': 9.5, 'Historia': 7.0, 'Ciencias': 9.5, 'Literatura': 8.0}
nota_buscada = 9.5
materias_con_nota_alta = []

for materia, nota in calificaciones.items():
    if nota == nota_buscada:
        materias_con_nota_alta.append(materia)

if materias_con_nota_alta:
    print(f"Las materias con nota {nota_buscada} son: {', '.join(materias_con_nota_alta)}")
else:
    print(f"Ninguna materia tiene una nota de {nota_buscada}")
```

Resultado:

```
Las materias con nota 9.5 son: Matemáticas, Ciencias
```

Ejercicio 1: Analizador de Frecuencia de Palabras

Enunciado:

Crea una función llamada `analizar_frecuencia` que reciba un texto (string) y devuelva un diccionario donde las claves sean las palabras del texto y los valores sean la cantidad de veces que cada palabra aparece. Ignora mayúsculas y minúsculas y los signos de puntuación básicos (`,` , `.`).

Ejemplo:

```
texto_ejemplo = "Este es un texto de ejemplo. Un texto simple para probar el ejemplo."
frecuencias = analizar_frecuencia(texto_ejemplo)
print(frecuencias)
# Salida esperada (el orden puede variar):
# {'este': 1, 'es': 1, 'un': 2, 'texto': 2, 'de': 1, 'ejemplo': 2, 'simple': 1, 'para': 1, 'probar': 1, 'el': 1}
```

Pista: Deberás procesar el string, dividirlo en palabras y luego iterar sobre esa lista de palabras para construir el diccionario.

Ejercicio 2: Invertir un Diccionario

Enunciado:
Escribe una función `invertir_diccionario` que reciba un diccionario y devuelva uno nuevo donde las claves sean los valores del diccionario original y los valores sean las claves originales.

Restricción: Asume que todos los valores del diccionario original son únicos.

Ejemplo:

```
capitales = {'Argentina': 'Buenos Aires', 'España': 'Madrid', 'Colombia': 'Bogotá'}
ciudades = invertir_diccionario(capitales)
print(ciudades)
# Salida esperada:
# {'Buenos Aires': 'Argentina', 'Madrid': 'España', 'Bogotá': 'Colombia'}
```

Pista: Itera sobre los `items` del diccionario original para construir el nuevo.

Ejercicio 3: Combinar Inventarios

Enunciado:

Escribe una función `combinar_inventarios` que reciba dos diccionarios (inventarios de dos tiendas). La función debe devolver un nuevo diccionario que represente el inventario combinado. Si un producto existe en ambas tiendas, sus cantidades deben sumarse.

Ejemplo:

```
tienda1 = {'manzanas': 50, 'naranjas': 35, 'plátanos': 40}
tienda2 = {'manzanas': 20, 'peras': 30, 'naranjas': 25}
inventario_total = combinar_inventarios(tienda1, tienda2)
print(inventario_total)
# Salida esperada:
# {'manzanas': 70, 'naranjas': 60, 'plátanos': 40, 'peras': 30}
```