11_diccionarios

January 6, 2025

Lenguaje de Programación

Universidad Politécnica Salesiana

Ingeniería en Ciencias de la Computación

Programación

Autores del Material

- **Instructor**: Ing. Pablo Torres
- **Contacto**: ptorresp@ups.edu.ec

A. Tema: Diccionarios en Python

Descripción General

En este tema, exploraremos dos de las estructuras de datos más poderosas y versátiles en Python: **diccionarios**. Los diccionarios permiten almacenar pares clave-valor, ofreciendo una manera eficiente de acceder y gestionar datos asociados. Por otro lado. Aprenderemos a crear, manipular y aplicar estas estructuras en diversos contextos de programación.

Objetivos de Aprendizaje

- Comprender la estructura y uso de diccionarios en Python.
- Aprender a crear y acceder a elementos en diccionarios.
- Realizar operaciones básicas y avanzadas con diccionarios.
- Aplicar métodos específicos para manipular diccionarios de manera eficiente.
- Resolver problemas prácticos utilizando diccionarios.
- Comparar y contrastar diccionarios con otras estructuras de datos como listas y tuplas.

---

1. Introducción a los Diccionarios

1.1. ¿Qué es un Diccionario?

Un **diccionario** en Python es una colección desordenada, mutable e indexada de pares clave-valor. Cada elemento en un diccionario tiene una **clave única** y un **valor** asociado. Los diccionarios son

ideales para almacenar datos que tienen una relación directa entre sí, como información de contacto, configuraciones, o cualquier otro tipo de datos que puedan ser identificados por una clave.

1.2. Creación de Diccionarios

Puedes crear diccionarios utilizando llaves `{}` y separando las claves de los valores con dos puntos `:`. También puedes utilizar la función `dict()` para crear diccionarios.

```
# Diccionario vacío

diccionario_vacio = {}



# Diccionario con elementos

estudiante = {

    "nombre": "Juan",

    "edad": 21,

    "carrera": "Ingeniería en Ciencias de la Computación"

}



print(estudiante)

# Output: {'nombre': 'Juan', 'edad': 21, 'carrera': 'Ingeniería en Ciencias de la Computación'



# Usando la función dict()

persona = dict(nombre="Ana", edad=22, ciudad="Quito")

print(persona)

# Output: {'nombre': 'Ana', 'edad': 22, 'ciudad': 'Quito'}
```

1.3. Acceso a Elementos en un Diccionario

Puedes acceder a los valores de un diccionario utilizando sus claves dentro de corchetes `[]`. También es posible utilizar el método `get()` para acceder a los valores, lo que evita errores si la clave no existe.

```
estudiante = {
```

```
    "nombre": "Juan",
    "edad": 21,
    "carrera": "Ingeniería en Sistemas"
}

# Acceder al valor de una clave específica
print(estudiante["nombre"])  # Output: Juan

# Usando el método get()
print(estudiante.get("edad"))  # Output: 21
print(estudiante.get("direccion", "No disponible"))  # Output: No disponible
```

1.4. Modificación de Elementos

Los diccionarios son **mutables**, lo que significa que puedes modificar, añadir o eliminar pares clave-valor después de su creación.

```
estudiante = {
    "nombre": "Juan",
    "edad": 21,
    "carrera": "Ingeniería en Sistemas"
}

# Actualizar el valor de una clave existente
estudiante["edad"] = 22
print(estudiante["edad"])  # Output: 22
```

```python
# Añadir una nueva clave-valor
estudiante["semestre"] = 5
print(estudiante)
# Output: {'nombre': 'Juan', 'edad': 22, 'carrera': 'Ingeniería en Sistemas', 'semestre': 5}

# Eliminar una clave-valor
del estudiante["semestre"]
print(estudiante)
# Output: {'nombre': 'Juan', 'edad': 22, 'carrera': 'Ingeniería en Sistemas'}
```

3. Operaciones Básicas con Diccionarios

3.1. Métodos Comunes de Diccionarios

- `keys()`: Retorna una vista de todas las claves en el diccionario.
- `values()`: Retorna una vista de todos los valores en el diccionario.
- `items()`: Retorna una vista de pares clave-valor.
- `get()`: Retorna el valor asociado a una clave, sin causar error si la clave no existe.
- `pop()`: Elimina una clave específica y retorna su valor.
- `clear()`: Elimina todos los elementos del diccionario.
- `update()`: Actualiza el diccionario con pares clave-valor de otro diccionario o iterable.

```python
estudiante = {
    "nombre": "Juan",
    "edad": 22,
    "carrera": "Ingeniería en Sistemas"
}

# Obtener todas las claves
print(estudiante.keys())  # Output: dict_keys(['nombre', 'edad', 'carrera'])
```

```
# Obtener todos los valores
print(estudiante.values())  # Output: dict_values(['Juan', 22, 'Ingeniería en Sistemas'])

# Obtener pares clave-valor
print(estudiante.items())
# Output: dict_items([('nombre', 'Juan'), ('edad', 22), ('carrera', 'Ingeniería en Sistemas')]

# Usar get()
print(estudiante.get("nombre"))  # Output: Juan
print(estudiante.get("direccion", "No disponible"))  # Output: No disponible

# Usar update()
nuevo_info = {"semestre": 5, "edad": 23}
estudiante.update(nuevo_info)
print(estudiante)
# Output: {'nombre': 'Juan', 'edad': 23, 'carrera': 'Ingeniería en Sistemas', 'semestre': 5}
```

4. Iteración sobre Diccionarios

4.1. Iterar sobre Diccionarios

Puedes iterar sobre las claves, valores o pares clave-valor de un diccionario.

```
estudiante = {
    "nombre": "Juan",
    "edad": 23,
    "carrera": "Ingeniería en Sistemas"
```

```
}


# Iterar sobre claves
for clave in estudiante:
    print(clave)
# Output:
# nombre
# edad
# carrera


# Iterar sobre valores
for valor in estudiante.values():
    print(valor)
# Output:
# Juan
# 23
# Ingeniería en Sistemas


# Iterar sobre pares clave-valor
for clave, valor in estudiante.items():
    print(f"{clave}: {valor}")
# Output:
# nombre: Juan
# edad: 23
```

```
# carrera: Ingeniería en Sistemas
```

B. Ejercicios Prácticos

Ejercicio 1: Crear y Manipular un Diccionario de Estudiantes

Descripción: Crea un diccionario que almacene información sobre 3 estudiantes, incluyendo su nombre, edad y lista de cursos. Luego, realiza las siguientes operaciones:

1. Añade un nuevo estudiante al diccionario.
2. Actualiza la edad de un estudiante existente.
3. Elimina el tercer estudiante del diccionario.
4. Muestra toda la información de los estudiantes.

Solución:

```
def manipular_diccionarios():

    # Crear el diccionario de estudiantes

    estudiantes = {

        "s001": {"nombre": "Juan", "edad": 21, "cursos": ["Python", "Matemáticas"]},

        "s002": {"nombre": "Ana", "edad": 22, "cursos": ["Java", "Física"]},

        "s003": {"nombre": "Luis", "edad": 20, "cursos": ["C++", "Química"]}

    }



    # Añadir un nuevo estudiante

    estudiantes["s004"] = {"nombre": "María", "edad": 23, "cursos": ["JavaScript", "Biología"]]



    # Actualizar la edad de un estudiante

    estudiantes["s002"]["edad"] = 23



    # Eliminar un estudiante

    del estudiantes["s003"]
```

```
    # Mostrar la información de los estudiantes
    for id_estudiante, info in estudiantes.items():
        print(f"ID: {id_estudiante}")
        for clave, valor in info.items():
            print(f"  {clave.capitalize()}: {valor}")
        print()

# Ejecutar la función
manipular_diccionarios()
```

Salida Esperada:

```
ID: s001
  Nombre: Juan
  Edad: 21
  Cursos: ['Python', 'Matemáticas']

ID: s002
  Nombre: Ana
  Edad: 23
  Cursos: ['Java', 'Física']

ID: s004
  Nombre: María
  Edad: 23
```

```
  Cursos: ['JavaScript', 'Biología']
```

Ejercicio 2: Contar Frecuencia de Palabras usando Diccionarios

Descripción: Crea una función que tome una lista de palabras ingresada por el usuario y retorne un diccionario con la frecuencia de cada palabra.

Solución:

```
def contar_frecuencia_palabras():

    # Solicitar al usuario una lista de palabras

    entrada = input("Ingrese una lista de palabras separadas por espacios: ")

    lista_palabras = entrada.split()



    # Crear el diccionario de frecuencia

    frecuencia = {}

    for palabra in lista_palabras:

        if palabra in frecuencia:

            frecuencia[palabra] += 1

        else:

            frecuencia[palabra] = 1



    # Mostrar la frecuencia de cada palabra

    print("Frecuencia de palabras:")

    for palabra, cuenta in frecuencia.items():

        print(f"{palabra}: {cuenta}")



# Ejecutar la función

contar_frecuencia_palabras()
```

Salida Esperada:

```
Ingrese una lista de palabras separadas por espacios: python java python c++ python

Frecuencia de palabras:

python: 3

java: 1

c++: 1
```

C. Conclusión

Los **diccionarios** y son estructuras de datos fundamentales en Python que ofrecen flexibilidad y eficiencia para almacenar y manipular datos de diferentes maneras. Los diccionarios son ideales para asociar claves con valores, permitiendo un acceso rápido y organizado a la información. Por otro lado.

Recomendaciones para los Estudiantes:

- **Practicar la Creación y Manipulación de Diccionarios**: Intenta crear diccionarios con diferentes tipos de datos y practica los métodos para añadir, eliminar y acceder a elementos.
- **Realizar Operaciones Variadas**: Implementa operaciones básicas y avanzadas como unión, intersección, y métodos como `get()`, `update()` en diccionarios.
- **Resolver Ejercicios Prácticos**: Completa los ejercicios propuestos y busca otros adicionales para aplicar tus conocimientos en situaciones reales.
- **Leer la Documentación Oficial**: Consulta la documentación oficial de Python sobre diccionarios
- **Comparar con Otras Estructuras de Datos**: Entiende cuándo es más apropiado usar un diccionario lugar de listas o tuplas para optimizar el rendimiento y la seguridad de tus datos.

¡Continúa practicando y fortaleciendo tus habilidades en Python! Dominar diccionarios te permitirá manejar datos de manera más eficiente y resolver problemas complejos con mayor facilidad.

Recomendaciones para los Estudiantes*

- **Practicar Regularmente**: La práctica constante es clave para dominar el manejo de diccionarios. Realiza los ejercicios propuestos y crea tus propios ejemplos.
- **Documentar tu Código**: Utiliza comentarios para explicar la lógica detrás de tus funciones y operaciones. Esto te ayudará a ti y a otros a entender mejor tu código.
- **Explorar Métodos Avanzados**: Investiga y experimenta con métodos adicionales de diccionarios para ampliar tus habilidades y mejorar la eficiencia de tus programas.
- **Optimizar Funciones**: Revisa tus soluciones para identificar oportunidades de optimización en términos de eficiencia y legibilidad.

- **Comparar con Otras Estructuras de Datos**: Entiende cuándo es más apropiado usar un diccionario en lugar de listas o tuplas para optimizar el rendimiento y la seguridad de tus datos.
- **Buscar Recursos Adicionales**: Consulta la documentación oficial de Python y otros recursos en línea para profundizar en tu conocimiento sobre diccionarios.

0.1 D. Recursos Adicionales

- Documentación Oficial de Python - Estructuras de Decisión
- Tutorial de Estructuras de Decisión en Real Python
- Ejemplos de Estructuras de Decisión en W3Schools

0.2 E. Referencias

1. Van Rossum, G. (1991). *Python Tutorial*. Python Software Foundation.
2. Lutz, M. (2013). *Learning Python*. O'Reilly Media.
3. Sweigart, A. (2015). *Automate the Boring Stuff with Python*. No Starch Press.
