

Diccionarios y Tuplas en Python

1. Introducción a los diccionarios

1.1 ¿Qué son los diccionarios?

Un **diccionario** es una colección **desordenada** de pares **clave-valor**. A diferencia de las listas que acceden por índice, los diccionarios acceden por clave, lo que los hace perfectos para almacenar información asociada.

En la vida cotidiana usamos diccionarios constantemente:

- "El teléfono de Carlos es 123456789"
- "El precio del producto 'Laptop' es \$1000"
- "El email de Ana es ana@example.com"

En programación, los diccionarios nos permiten:

- Almacenar datos relacionados con claves significativas.
- Acceder a datos rápidamente por clave.
- Representar objetos del mundo real.
- Crear estructuras complejas.

1.2 Crear un diccionario

```
# Diccionario simple
estudiante = {
    "nombre": "Carlos",
    "edad": 20,
    "carrera": "Ingeniería"
}

# Diccionario vacío
diccionario_vacio = {}

# Diccionario con diferentes tipos
datos_mixtos = {
    "nombre": "Ana",
    "edad": 22,
    "es_activo": True,
    "promedio": 3.85
}

# Diccionario anidado
```

```
persona = {  
    "nombre": "Juan",  
    "contacto": {  
        "email": "juan@example.com",  
        "telefono": "555-1234"  
    }  
}
```

1.3 Ejemplo introductorio

```
# Crear un diccionario de un contacto  
contacto = {  
    "nombre": "María",  
    "email": "maria@example.com",  
    "telefono": "555-5678",  
    "ciudad": "Quito"  
}  
  
# Acceder a valores  
print(contacto["nombre"]) # María  
print(contacto["email"])  # maria@example.com  
  
# Modificar un valor  
contacto["telefono"] = "555-9999"  
print(contacto) # {'nombre': 'María', 'email': 'maria@example.com', ...}  
  
# Agregar una nueva clave  
contacto["edad"] = 28  
print(contacto) # Ahora incluye 'edad': 28  
  
# Saber cuántos pares clave-valor hay  
print(len(contacto)) # 5
```

2. Acceso a elementos en diccionarios

2.1 Acceso por clave

```
alumno = {  
    "nombre": "Pedro",  
    "edad": 21,  
    "carrera": "Sistemas"  
}  
  
# Acceder a valores  
print(alumno["nombre"]) # Pedro  
print(alumno["edad"])   # 21
```

```
print(alumno["carrera"]) # Sistemas

# Intentar acceder a una clave inexistente
print(alumno["ciudad"]) # ✖ KeyError
```

2.2 Método get() - Acceso seguro

El método `get()` retorna `None` si la clave no existe, en lugar de lanzar error:

```
alumno = {"nombre": "Pedro", "edad": 21}

# Con get() - seguro
print(alumno.get("nombre")) # Pedro
print(alumno.get("ciudad")) # None
print(alumno.get("ciudad", "No especificada")) # No especificada
```

2.3 Método keys() - Obtener todas las claves

```
producto = {
    "nombre": "Laptop",
    "precio": 1000,
    "stock": 5,
    "marca": "Dell"
}

# Obtener todas las claves
claves = producto.keys()
print(claves) # dict_keys(['nombre', 'precio', 'stock', 'marca'])

# Iterar sobre las claves
for clave in producto.keys():
    print(clave)
```

2.4 Método values() - Obtener todos los valores

```
producto = {
    "nombre": "Laptop",
    "precio": 1000,
    "stock": 5
}

# Obtener todos los valores
valores = producto.values()
print(valores) # dict_values(['Laptop', 1000, 5])

# Iterar sobre los valores
```

```
for valor in producto.values():  
    print(valor)
```

2.5 Método items() - Obtener pares clave-valor

```
estudiante = {  
    "nombre": "Ana",  
    "edad": 22,  
    "promedio": 3.85  
}  
  
# Obtener pares clave-valor  
pares = estudiante.items()  
print(pares)  # dict_items([('nombre', 'Ana'), ('edad', 22), ('promedio', 3.85)])  
  
# Iterar sobre pares  
for clave, valor in estudiante.items():  
    print(f"{clave}: {valor}")
```

3. Modificación de diccionarios

3.1 Agregar elementos

```
contacto = {"nombre": "Carlos"}  
  
# Agregar nueva clave-valor  
contacto["telefono"] = "555-1234"  
contacto["email"] = "carlos@example.com"  
  
print(contacto)  # {'nombre': 'Carlos', 'telefono': '555-1234', 'email': 'carlos@example.com'}
```

3.2 Modificar elementos

```
usuario = {  
    "nombre": "Juan",  
    "edad": 25,  
    "ciudad": "Quito"  
}  
  
# Modificar valores existentes  
usuario["edad"] = 26  
usuario["ciudad"] = "Guayaquil"
```

```
print(usuario) # {'nombre': 'Juan', 'edad': 26, 'ciudad': 'Guayaquil'}
```

3.3 Eliminar por clave: pop()

```
producto = {  
    "nombre": "Mouse",  
    "precio": 25,  
    "marca": "Logitech"  
}  
  
# Eliminar y obtener el valor  
precio_eliminado = producto.pop("precio")  
print(precio_eliminado) # 25  
print(producto) # {'nombre': 'Mouse', 'marca': 'Logitech'}  
  
# Con valor por defecto si no existe  
marca_eliminada = producto.pop("color", "No especificado")  
print(marca_eliminada) # No especificado
```

3.4 Eliminar último par: popitem()

```
diccionario = {"a": 1, "b": 2, "c": 3}  
  
# Eliminar el último par (en Python 3.7+)  
clave, valor = diccionario.popitem()  
print(f"Eliminado: {clave} = {valor}") # Eliminado: c = 3  
print(diccionario) # {'a': 1, 'b': 2}
```

3.5 Limpiar diccionario: clear()

```
diccionario = {"a": 1, "b": 2, "c": 3}  
  
diccionario.clear()  
print(diccionario) # {}  
print(len(diccionario)) # 0
```

3.6 Actualizar con otro diccionario: update()

```
usuario = {  
    "nombre": "Pedro",  
    "edad": 25  
}  
  
# Agregar múltiples pares
```

```
usuario.update({  
    "ciudad": "Latacunga",  
    "carrera": "Ingeniería"  
})
```

```
print(usuario) # {'nombre': 'Pedro', 'edad': 25, 'ciudad': 'Latacunga', 'carrera': 'Ingenierí
```

3.7 Método.setdefault() - Agregar con valor por defecto

```
diccionario = {"nombre": "Ana"}  
  
# Si no existe, agrega con valor por defecto  
resultado = diccionario.setdefault("edad", 25)  
print(resultado) # 25  
print(diccionario) # {'nombre': 'Ana', 'edad': 25}  
  
# Si ya existe, no cambia  
resultado = diccionario.setdefault("edad", 30)  
print(resultado) # 25 (no cambió)
```

4. Copiar diccionarios

4.1 Referencia vs copia

```
# Referencia (comparte contenido)  
diccionario1 = {"a": 1, "b": 2}  
diccionario2 = diccionario1  
diccionario2["a"] = 10  
print(diccionario1) # {'a': 10, 'b': 2} ← Cambió!  
  
# Copia (independiente)  
diccionario1 = {"a": 1, "b": 2}  
diccionario2 = diccionario1.copy()  
diccionario2["a"] = 10  
print(diccionario1) # {'a': 1, 'b': 2} ← No cambió
```

5. Diccionarios anidados

5.1 Crear diccionarios anidados

```
escuela = {  
    "nombre": "Colegio Central",
```

```
"director": {
    "nombre": "Dr. García",
    "email": "garcia@escuela.com"
},
"estudiantes": [
    {"nombre": "Ana", "grado": 10},
    {"nombre": "Carlos", "grado": 11}
]
}

# Acceder a elementos anidados
print(escuela["nombre"])           # Colegio Central
print(escuela["director"]["nombre"]) # Dr. García
print(escuela["estudiantes"][0]["nombre"]) # Ana
```

5.2 Modificar elementos anidados

```
persona = {
    "nombre": "Juan",
    "contacto": {
        "email": "juan@example.com",
        "telefono": "555-1234"
    }
}

# Modificar valor anidado
persona["contacto"]["email"] = "juan.nuevo@example.com"

# Agregar a diccionario anidado
persona["contacto"]["ciudad"] = "Quito"

print(persona)
```

6. Iteración sobre diccionarios

6.1 Iterar sobre claves

```
producto = {
    "nombre": "Teclado",
    "precio": 50,
    "stock": 10
}

for clave in producto:
    print(clave) # nombre, precio, stock
```

6.2 Iterar sobre valores

```
producto = {  
    "nombre": "Teclado",  
    "precio": 50,  
    "stock": 10  
}  
  
for valor in producto.values():  
    print(valor)  # Teclado, 50, 10
```

6.3 Iterar sobre pares clave-valor

```
estudiante = {  
    "nombre": "María",  
    "edad": 20,  
    "carrera": "Sistemas"  
}  
  
for clave, valor in estudiante.items():  
    print(f"{clave}: {valor}")
```

7. Introducción a las tuplas

7.1 ¿Qué son las tuplas?

Una **tupla** es una colección **ordenada e inmutable** de elementos. A diferencia de las listas, las tuplas no se pueden modificar después de su creación, lo que las hace más seguras y eficientes para datos constantes.

7.2 Crear tuplas

```
# Tupla simple  
coordenadas = (10, 20, 30)  
  
# Tupla de strings  
colores = ("rojo", "verde", "azul")  
  
# Tupla vacía  
tupla_vacia = ()  
  
# Tupla con un elemento (nota la coma)  
tupla_uno = (42,)  # ¡Importante la coma!
```

```
# Tupla con diferentes tipos
datos = (1, "hola", 3.14, True)

# Tupla anidada
matriz = ((1, 2), (3, 4), (5, 6))
```

7.3 Acceso a elementos

```
colores = ("rojo", "verde", "azul")

# Acceso por índice
print(colores[0])    # rojo
print(colores[1])    # verde
print(colores[-1])   # azul

# Slicing
print(colores[0:2])  # ('rojo', 'verde')
print(colores[::-1]) # ('azul', 'verde', 'rojo')
```

7.4 Inmutabilidad - No se puede modificar

```
tupla = (1, 2, 3)

# ❌ Error: no se puede modificar
tupla[0] = 10 # TypeError

# ❌ Error: no se puede agregar
tupla.append(4) # AttributeError

# ✅ Puedes crear una nueva tupla
tupla = tupla + (4, 5)
print(tupla) # (1, 2, 3, 4, 5)
```

7.5 Métodos de tuplas

```
numeros = (1, 2, 2, 3, 3, 3)

# count() - contar ocurrencias
print(numeros.count(3)) # 3
print(numeros.count(1)) # 1

# index() - encontrar índice
print(numeros.index(2)) # 1
print(numeros.index(3)) # 3
```

8. Tuplas como claves de diccionarios

Debido a que las tuplas son inmutables, se pueden usar como claves en diccionarios, mientras que las listas no:

```
# ✅ Tupla como clave (válido)
localizaciones = {
    (10, 20): "Punto A",
    (30, 40): "Punto B",
    (50, 60): "Punto C"
}

print(localizaciones[(10, 20)]) # Punto A

# ❌ Lista como clave (error)
diccionario = {
    [1, 2]: "valor" # TypeError: unhashable type
}
```

9. Diferencias: Listas vs Tuplas vs Diccionarios

Característica	Lista	Tupla	Diccionario
Ordenada	Sí	Sí	No (3.7+ mantiene orden)
Mutable	Sí	No	Sí
Acceso	Por índice	Por índice	Por clave
Método acceso	lista[0]	tupla[0]	dict["clave"]
Como clave dict	No	Sí	-
Eficiencia	Buena	Mejor	Muy buena

10. Dict Comprehension

10.1 ¿Qué es dict comprehension?

Similar a list comprehension, permite crear diccionarios de forma compacta:

```
{clave: valor for elemento in iterable if condicion}
```

10.2 Ejemplos

```
# Crear diccionario de números al cuadrado
numeros = [1, 2, 3, 4, 5]
cuadrados = {n: n**2 for n in numeros}
print(cuadrados) # {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# Filtrar diccionario
diccionario_original = {"a": 1, "b": 2, "c": 3, "d": 4}
pares = {k: v for k, v in diccionario_original.items() if v % 2 == 0}
print(pares) # {'b': 2, 'd': 4}

# Invertir claves y valores
diccionario = {"nombre": "Ana", "edad": "22", "ciudad": "Quito"}
invertido = {v: k for k, v in diccionario.items()}
print(invertido) # {'Ana': 'nombre', '22': 'edad', 'Quito': 'ciudad'}
```

11. Ejercicio guiado: Sistema de contactos

Crearemos un programa que almacene y gestione contactos con diccionarios:

```
# Sistema de contactos

print("=" * 50)
print("SISTEMA DE CONTACTOS".center(50))
print("=" * 50)
print()

contactos = {}

while True:
    print("-" * 50)
    print("MENÚ CONTACTOS")
    print("-" * 50)
    print("1. Agregar contacto")
    print("2. Ver todos los contactos")
    print("3. Buscar contacto")
    print("4. Eliminar contacto")
    print("5. Actualizar contacto")
    print("6. Salir")
    print("-" * 50)

    opcion = input("Elige una opción (1-6): ")

    # OPCIÓN 1: Agregar contacto
    if opcion == "1":
        print()
        nombre = input("Nombre del contacto: ")
```

```

if nombre in contactos:
    print(f"❌ El contacto '{nombre}' ya existe")
else:
    email = input("Email: ")
    telefono = input("Teléfono: ")
    ciudad = input("Ciudad: ")

    contactos[nombre] = {
        "email": email,
        "telefono": telefono,
        "ciudad": ciudad
    }
    print(f"✅ Contacto '{nombre}' agregado")
print()

# OPCIÓN 2: Ver todos
elif opcion == "2":
    print()
    if len(contactos) == 0:
        print("🚫 No hay contactos")
    else:
        print("📋 CONTACTOS")
        for nombre, info in contactos.items():
            print(f"\n👤 {nombre}")
            print(f"    Email: {info['email']}")
            print(f"    Teléfono: {info['telefono']}")
            print(f"    Ciudad: {info['ciudad']}")
        print()

# OPCIÓN 3: Buscar contacto
elif opcion == "3":
    print()
    nombre = input("Nombre a buscar: ")

    if nombre in contactos:
        print(f"✅ Contacto encontrado:")
        info = contactos[nombre]
        print(f"    Email: {info['email']}")
        print(f"    Teléfono: {info['telefono']}")
        print(f"    Ciudad: {info['ciudad']}")
    else:
        print(f"❌ Contacto '{nombre}' no encontrado")
    print()

# OPCIÓN 4: Eliminar contacto
elif opcion == "4":
    print()
    nombre = input("Nombre a eliminar: ")

    if nombre in contactos:
        del contactos[nombre]
        print(f"✅ Contacto '{nombre}' eliminado")

```

```
else:
    print(f"❌ Contacto '{nombre}' no encontrado")
    print()

# OPCIÓN 5: Actualizar contacto
elif opcion == "5":
    print()
    nombre = input("Nombre a actualizar: ")

    if nombre in contactos:
        print("¿Qué deseas actualizar?")
        print("1. Email")
        print("2. Teléfono")
        print("3. Ciudad")

        sub_opcion = input("Opción (1-3): ")

        if sub_opcion == "1":
            nuevo_email = input("Nuevo email: ")
            contactos[nombre]["email"] = nuevo_email
            print("✅ Email actualizado")
        elif sub_opcion == "2":
            nuevo_tel = input("Nuevo teléfono: ")
            contactos[nombre]["telefono"] = nuevo_tel
            print("✅ Teléfono actualizado")
        elif sub_opcion == "3":
            nueva_ciudad = input("Nueva ciudad: ")
            contactos[nombre]["ciudad"] = nueva_ciudad
            print("✅ Ciudad actualizada")
    else:
        print(f"❌ Contacto '{nombre}' no encontrado")
        print()

# OPCIÓN 6: Salir
elif opcion == "6":
    print()
    print("✅ ¡Hasta luego!")
    break

else:
    print()
    print("❌ Opción no válida")
    print()
```

12. Ejercicio de evaluación

12.1 Enunciado

Crea un programa que gestione un **registro de estudiantes y sus calificaciones** con las siguientes funcionalidades:

1. **Agregar estudiante:** Nombre, carrera, teléfono
2. **Agregar calificación:** Elegir estudiante y agregar nota (asignatura y calificación)
3. **Ver estudiante:** Mostrar todos sus datos y calificaciones
4. **Calcular promedio:** Calcular promedio de un estudiante
5. **Estudiante con mejor promedio:** Mostrar cuál tiene el mejor promedio
6. **Eliminar estudiante:** Remover del registro
7. **Salir**

Estructura de datos sugerida:

```
estudiantes = {  
    "Carlos": {  
        "carrera": "Sistemas",  
        "telefono": "555-1234",  
        "calificaciones": {  
            "Python": 18,  
            "Bases de Datos": 16  
        }  
    }  
}
```

12.2 Pistas

- Usa diccionarios anidados para estructurar los datos
- Para agregar calificación, accede a `estudiantes[nombre]["calificaciones"][asignatura] = nota`
- Para calcular promedio: `sum(calificaciones.values()) / len(calificaciones)`
- Usa `max()` con parámetro `key` para encontrar el mejor promedio

13. Solución del ejercicio de evaluación

```
# Registro de estudiantes y calificaciones  
  
print("=" * 60)  
print("SISTEMA DE REGISTRO DE ESTUDIANTES".center(60))  
print("=" * 60)  
print()  
  
estudiantes = {}  
  
while True:
```

```

print("-" * 60)
print("MENÚ PRINCIPAL")
print("-" * 60)
print("1. Agregar estudiante")
print("2. Agregar calificación")
print("3. Ver estudiante")
print("4. Calcular promedio")
print("5. Mejor promedio general")
print("6. Eliminar estudiante")
print("7. Salir")
print("-" * 60)

opcion = input("Elige una opción (1-7): ")

# OPCIÓN 1: Agregar estudiante
if opcion == "1":
    print()
    nombre = input("Nombre del estudiante: ")

    if nombre in estudiantes:
        print(f"❌ El estudiante '{nombre}' ya existe")
    else:
        carrera = input("Carrera: ")
        telefono = input("Teléfono: ")

        estudiantes[nombre] = {
            "carrera": carrera,
            "telefono": telefono,
            "calificaciones": {}
        }
        print(f"✅ Estudiante '{nombre}' agregado")
    print()

# OPCIÓN 2: Agregar calificación
elif opcion == "2":
    print()
    nombre = input("Nombre del estudiante: ")

    if nombre not in estudiantes:
        print(f"❌ Estudiante '{nombre}' no encontrado")
    else:
        asignatura = input("Asignatura: ")
        try:
            calificacion = float(input("Calificación (0-20): "))

            if calificacion < 0 or calificacion > 20:
                print(f"❌ La calificación debe estar entre 0 y 20")
            else:
                estudiantes[nombre]["calificaciones"][asignatura] = calificacion
                print(f"✅ Calificación agregada a '{nombre}'")
        except ValueError:
            print(f"❌ Ingresa un número válido")
    print()

```

```

# OPCIÓN 3: Ver estudiante
elif opcion == "3":
    print()
    nombre = input("Nombre del estudiante: ")

    if nombre not in estudiantes:
        print(f"❌ Estudiante '{nombre}' no encontrado")
    else:
        info = estudiantes[nombre]
        print(f"\n👤 {nombre}")
        print(f"    Carrera: {info['carrera']}")
        print(f"    Teléfono: {info['telefono']}")
        print(f"    Calificaciones:")

        if len(info['calificaciones']) == 0:
            print("        (Sin calificaciones)")
        else:
            for asignatura, nota in info['calificaciones'].items():
                print(f"        - {asignatura}: {nota}")

    print()

# OPCIÓN 4: Calcular promedio
elif opcion == "4":
    print()
    nombre = input("Nombre del estudiante: ")

    if nombre not in estudiantes:
        print(f"❌ Estudiante '{nombre}' no encontrado")
    else:
        calificaciones = estudiantes[nombre]["calificaciones"]

        if len(calificaciones) == 0:
            print(f"❌ {nombre} no tiene calificaciones")
        else:
            promedio = sum(calificaciones.values()) / len(calificaciones)
            print(f"📊 Promedio de {nombre}: {promedio:.2f}")

    print()

# OPCIÓN 5: Mejor promedio general
elif opcion == "5":
    print()
    if len(estudiantes) == 0:
        print("❌ No hay estudiantes registrados")
    else:
        mejor_estudiante = None
        mejor_promedio = -1

        for nombre, info in estudiantes.items():
            calificaciones = info["calificaciones"]

            if len(calificaciones) > 0:
                promedio = sum(calificaciones.values()) / len(calificaciones)

```

```

        if promedio > mejor_promedio:
            mejor_promedio = promedio
            mejor_estudiante = nombre

    if mejor_estudiante is None:
        print("❌ No hay estudiantes con calificaciones")
    else:
        print(f"🏆 Mejor promedio: {mejor_estudiante} con {mejor_promedio:.2f}")
    print()

# OPCIÓN 6: Eliminar estudiante
elif opcion == "6":
    print()
    nombre = input("Nombre a eliminar: ")

    if nombre not in estudiantes:
        print(f"❌ Estudiante '{nombre}' no encontrado")
    else:
        del estudiantes[nombre]
        print(f"✅ Estudiante '{nombre}' eliminado")
    print()

# OPCIÓN 7: Salir
elif opcion == "7":
    print()
    print("✅ ¡Hasta luego!")
    break

else:
    print()
    print("❌ Opción no válida")
    print()

```

14. Ejercicios adicionales

14.1 Crear índice de palabras

Crea un programa que cuente la frecuencia de palabras en un texto.

14.2 Carrito de compras con diccionarios

Sistema que guarde productos con cantidad y precio.

14.3 Convertidor de monedas

Diccionario con tasas de cambio, convertir entre monedas.

14.4 Agenda con tuplas

Almacenar eventos como tuplas (fecha, hora, descripción).

15. Tabla resumen de diccionarios y tuplas

Método	Descripción	Ejemplo
<code>dict.keys()</code>	Obtener claves	<code>dict.keys()</code>
<code>dict.values()</code>	Obtener valores	<code>dict.values()</code>
<code>dict.items()</code>	Obtener pares	<code>dict.items()</code>
<code>dict.get()</code>	Acceso seguro	<code>dict.get("clave", default)</code>
<code>dict.pop()</code>	Eliminar por clave	<code>dict.pop("clave")</code>
<code>dict.update()</code>	Fusionar	<code>dict.update(otro_dict)</code>
<code>dict.clear()</code>	Limpiar	<code>dict.clear()</code>
<code>len()</code>	Longitud	<code>len(dict)</code>
<code>tuple.count()</code>	Contar elementos	<code>tupla.count(elemento)</code>
<code>tuple.index()</code>	Encontrar índice	<code>tupla.index(elemento)</code>

16. Preguntas frecuentes PDF 5

P: ¿Cuál es la diferencia entre diccionarios y listas?

R: Las listas acceden por índice, los diccionarios por clave. Los diccionarios son más legibles para datos relacionados.

P: ¿Puedo modificar una tupla?

R: No, las tuplas son inmutables. Pero puedes crear una nueva tupla.

P: ¿Se pueden usar listas como clave de diccionario?

R: No, solo tuplas e inmutables. Las listas son mutables.

P: ¿Qué es dict comprehension?

R: Una forma compacta de crear diccionarios, similar a list comprehension.

Conclusión PDF 5

Felicidades, ahora entiendes:

- Diccionarios y operaciones clave-valor
- Tuplas e inmutabilidad
- Diccionarios anidados
- Dict comprehension
- Casos de uso prácticos