Comprensión de diccionarios

La comprensión de diccionarios (también conocida como "dict comprehensions" en inglés) es una característica en Python que permite crear diccionarios de forma concisa y eficiente mediante la aplicación de una expresión a pares clave-valor en una secuencia o mediante la iteración de una secuencia y la aplicación de una condición.

La sintaxis básica de una comprensión de diccionarios se asemeja a la de las comprensiones de listas, pero en lugar de crear una lista, se crea un diccionario. Aquí tienes la forma básica de una comprensión de diccionarios:

```
nuevo_diccionario = {clave: valor for elemento in secuencia}
```

- **clave** es la expresión que define la clave del diccionario.
- **valor** es la expresión que define el valor asociado a la clave.
- **elemento** es una variable que representa cada elemento en la secuencia de origen.

Además, puedes agregar una cláusula if para filtrar elementos según una condición:

```
nuevo_diccionario = {clave: valor for elemento in secuencia if condicion}
```

```
nueva_lista = [expresion for elemento in secuencia if condicion]
```

- **condicion** es una expresión booleana que determina si se incluye el par clave-valor en el diccionario.

La comprensión de diccionarios es útil para crear diccionarios de manera eficiente a partir de secuencias existentes, transformar datos en diccionarios y filtrar datos de acuerdo con ciertas condiciones. Es una característica versátil que ahorra tiempo y mejora la legibilidad del código.

GUIA EJERCICIOS BASICOS COMPRENSIÓN DE DICCIONARIOS

En todos los casos resolver utilizando una comprensión de diccionarios y la forma tradicional mediante bucle *for*

1. Dado un diccionario de estudiantes y sus calificaciones:

   ```
   notas = { "Ana": 8, "Juan": 7, "María": 9, "Carlos": 6, "Luisa": 5, "Pedro": 10, "Sofía": 8, "Pablo": 6, "Laura": 9, "Diego": 7 }
   ```

   Crea un nuevo diccionario donde las calificaciones donde solo se encuentren los alumnos que sacaron nota mayor o igual a 5.

2. Dado una lista de colores y sus valores hexadecimales

   colores_hex = {
   "Rojo": "#FF0000",
   "Verde": "#00FF00",
   "Azul": "#0000FF",
   "Amarillo": "#FFFF00",
   "Naranja": "#FFA500",
   "Violeta": "#8A2BE2",
   "Rosa": "#FFC0CB",

```
    "Gris": "#808080",
    "Cian": "#00FFFF",
    "Marrón": "#A52A2A"
}
```

Crea un nuevo diccionario donde los códigos hexadecimales sean las claves y los nombres de colores sean los valores.

3. Dado un diccionario de productos y sus precios:

```
productos_precios = { "Manzanas": 50.5, "Bananas": 30.10, "Naranjas": 20.0, "Uvas": 55.10, "Peras": 38.10, "Frutillas": 80.5, "Kiwi": 120.10, "Mangos": 50.5, "Sandía": 20.1, "Papayas": 32.5 }
```

   Crea un nuevo diccionario que aplique un descuento del 10% a cada producto.

4. Dado el siguiente texto, crear un diccionario donde clave es una letra y cada valor es la cantidad de veces que aparece la letra dentro del texto

```
zen_python = '''
El Zen de Python, por Tim Peters
Bello es mejor que feo.
Explícito es mejor que implícito.
Simple es mejor que complejo.
Complejo es mejor que complicado.
Plano es mejor que anidado.
Disperso es mejor que denso.
La legibilidad cuenta.
Los casos especiales no son tan especiales como para quebrantar las reglas.
Aunque lo práctico le gana a la pureza.
Errores nunca deberían pasar silenciosamente.
A menos que se silencien explícitamente.
Frente a la ambigüedad, evitar la tentación de adivinar.
Debería haber una, y preferiblemente solo una, manera obvia de hacerlo.
```

```
Aunque esa manera puede no ser obvia en un primer momento a menos que usted sea holandés.
Ahora es mejor que nunca.
Aunque nunca es a menudo mejor que *ahora* mismo.
Si la implementación es difícil de explicar, es una mala idea.
Si la implementación es fácil de explicar, puede que sea una buena idea.
Los espacios de nombres son una gran idea ¡Hagamos más de eso!
'''
```