**Dictionary Comprehension in Python**

Dictionary comprehension is a concise and elegant way to create dictionaries in Python. It allows you to generate a dictionary using a single line of code, similar to list comprehensions.

**Syntax**

{key\_expr: value\_expr for item in iterable if condition}

* key\_expr: Expression for the dictionary key.
* value\_expr: Expression for the dictionary value.
* iterable: A sequence (like a list, tuple, or range) to iterate over.
* condition (optional): A filtering condition to include only certain elements.

**Examples**

**1. Creating a Dictionary from a List**

numbers = [1, 2, 3, 4, 5]

squares = {num: num\*\*2 for num in numbers}

print(squares)

**Output:**

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

🔹 Here, each number in the list becomes a key, and its square is the corresponding value.

**2. Filtering with Dictionary Comprehension**

numbers = range(10)

even\_squares = {num: num\*\*2 for num in numbers if num % 2 == 0}

print(even\_squares)

**Output:**

{0: 0, 2: 4, 4: 16, 6: 36, 8: 64}

🔹 This only includes even numbers as keys.

**3. Swapping Keys and Values**

original\_dict = {'a': 1, 'b': 2, 'c': 3}

swapped\_dict = {value: key for key, value in original\_dict.items()}

print(swapped\_dict)

**Output:**

{1: 'a', 2: 'b', 3: 'c'}

🔹 This reverses the dictionary by making values the keys and vice versa.

**4. Using Dictionary Comprehension with zip()**

keys = ['name', 'age', 'city']

values = ['Alice', 25, 'New York']

person\_dict = {k: v for k, v in zip(keys, values)}

print(person\_dict)

**Output:**

{'name': 'Alice', 'age': 25, 'city': 'New York'}

🔹 The zip() function pairs the keys and values.

**5. Nested Dictionary Comprehension**

matrix = {i: {j: i\*j for j in range(1, 6)} for i in range(1, 4)}

print(matrix)

**Output:**

{

1: {1: 1, 2: 2, 3: 3, 4: 4, 5: 5},

2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10},

3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15}

}

🔹 This creates a dictionary of multiplication tables.

**When to Use Dictionary Comprehension?**

✅ When you need to create dictionaries in a concise and readable way.  
✅ When transforming or filtering data efficiently.  
✅ When readability is not sacrificed for brevity.

However, if the comprehension becomes too complex, consider using a standard for loop for better readability.