**Python Dictionaries: Comprehensive Guide with Examples & Interview Prep**

**Table of Contents**

1. [Introduction to Dictionaries](https://chatgpt.com/#introduction)
2. [Dictionary Operations](https://chatgpt.com/#operations)
3. [Dictionary Methods](https://chatgpt.com/#methods)
4. [Advanced Dictionary Concepts](https://chatgpt.com/#advanced)
5. [Lists and Tuples in Dictionaries](https://chatgpt.com/#lists-tuples)
6. [Interview Preparation](https://chatgpt.com/#interview)
7. [Practice Exercises](https://chatgpt.com/#exercises)

**1. Introduction to Dictionaries**

Dictionaries are Python's implementation of a data structure called an associative array (or hash table). They consist of key-value pairs.

**Characteristics:**

* **Unordered** (Python 3.7+ maintains insertion order)
* **Mutable** (can be changed after creation)
* **Keys must be unique and immutable** (strings, numbers, tuples)
* **Values** can be of any type

**Creating Dictionaries:**

# Empty dictionary

empty\_dict = {}

empty\_dict = dict()

# Dictionary with initial values

student = {'name': 'John', 'age': 25, 'courses': ['Math', 'CompSci']}

grades = dict(Alice=95, Bob=88, Charlie=92)

print(student)

print(grades)

**2. Dictionary Operations**

**Accessing Values:**

student = {'name': 'John', 'age': 25, 'courses': ['Math', 'CompSci']}

# Access using key

print(student['name']) # Output: John

# Access with get() - safer as it returns None if key doesn't exist

print(student.get('phone')) # Output: None

print(student.get('phone', 'Not Found')) # Output: Not Found

**Adding/Updating Items:**

# Adding a new key-value pair

student['phone'] = '555-5555'

print(student)

# Updating multiple values at once

student.update({'name': 'Jane', 'age': 26, 'phone': '555-1234'})

print(student)

**Removing Items:**

# Using del

del student['age']

print(student)

# Using pop() - returns the value

phone = student.pop('phone')

print(phone)

print(student)

# Clear all items

student.clear()

print(student) # Output: {}

# Delete entire dictionary

del student

**3. Dictionary Methods**

**Common Methods:**

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

# Get all keys

print(my\_dict.keys()) # dict\_keys(['a', 'b', 'c'])

# Get all values

print(my\_dict.values()) # dict\_values([1, 2, 3])

# Get all key-value pairs

print(my\_dict.items()) # dict\_items([('a', 1), ('b', 2), ('c', 3)])

# Length of dictionary

print(len(my\_dict)) # 3

# Check if key exists

print('a' in my\_dict) # True

print('z' in my\_dict) # False

**Dictionary Comprehensions:**

# Create a dictionary with squares of numbers

squares = {x: x\*x for x in range(1, 6)}

print(squares) # {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# Create from two lists

keys = ['a', 'b', 'c']

values = [1, 2, 3]

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

print(new\_dict) # {'a': 1, 'b': 2, 'c': 3}

**4. Advanced Dictionary Concepts**

**Default Dictionaries:**

from collections import defaultdict

# Default dictionary with list as default factory

dd = defaultdict(list)

dd['fruits'].append('apple')

dd['fruits'].append('banana')

dd['vegetables'].append('carrot')

print(dd) # defaultdict(<class 'list'>, {'fruits': ['apple', 'banana'], 'vegetables': ['carrot']})

**Ordered Dictionaries:**

from collections import OrderedDict

# Remember insertion order (useful in Python versions < 3.7)

od = OrderedDict()

od['a'] = 1

od['b'] = 2

od['c'] = 3

print(od) # OrderedDict([('a', 1), ('b', 2), ('c', 3)])

**Merging Dictionaries (Python 3.5+):**

dict1 = {'a': 1, 'b': 2}

dict2 = {'b': 3, 'c': 4}

# Using \*\* operator

merged = {\*\*dict1, \*\*dict2}

print(merged) # {'a': 1, 'b': 3, 'c': 4}

# Using | operator (Python 3.9+)

merged = dict1 | dict2

print(merged) # {'a': 1, 'b': 3, 'c': 4}

**5. Lists and Tuples in Dictionaries**

**Using Lists as Dictionary Values:**

# Dictionary with list values

student = {

'name': 'Alice',

'grades': [85, 92, 88, 95],

'courses': ['Math', 'Physics', 'Chemistry']

}

# Accessing list elements

print(student['grades'][1]) # 92

print(student['courses'][-1]) # Chemistry

# Modifying lists

student['grades'].append(90)

student['courses'].remove('Physics')

print(student)

**Using Tuples as Dictionary Keys:**

# Tuple as key (must contain only immutable elements)

coordinates = {

(35.6895, 139.6917): 'Tokyo',

(40.7128, -74.0060): 'New York',

(51.5074, -0.1278): 'London'

}

print(coordinates[(40.7128, -74.0060)]) # New York

# Adding new tuple key

coordinates[(48.8566, 2.3522)] = 'Paris'

print(coordinates)

**Dictionary with Tuple Values:**

# Storing multiple values per key

employees = {

'E001': ('John', 'Doe', 28, 'Developer'),

'E002': ('Jane', 'Smith', 32, 'Manager'),

'E003': ('Bob', 'Johnson', 45, 'Director')

}

# Accessing tuple elements

name, surname, age, position = employees['E001']

print(f"{name} {surname} is a {age}-year-old {position}")

**Nested Dictionaries with Lists and Tuples:**

university = {

'departments': {

'CS': {

'courses': [('CS101', 'Intro to Programming'), ('CS201', 'Data Structures')],

'professors': ['Dr. Smith', 'Dr. Johnson']

},

'Math': {

'courses': [('MATH101', 'Calculus I'), ('MATH202', 'Linear Algebra')],

'professors': ['Dr. Brown', 'Dr. Davis']

}

}

}

# Accessing nested data

print(university['departments']['CS']['courses'][0][1]) # Intro to Programming

**6. Interview Preparation**

**Common Dictionary Interview Questions:**

1. **How to merge two dictionaries?**
2. # Python 3.5+
3. dict1 = {'a': 1, 'b': 2}
4. dict2 = {'b': 3, 'c': 4}
5. merged = {\*\*dict1, \*\*dict2}
6. # Python 3.9+
7. merged = dict1 | dict2
8. **How to sort a dictionary by value?**
9. my\_dict = {'apple': 5, 'banana': 2, 'cherry': 8}
10. sorted\_dict = {k: v for k, v in sorted(my\_dict.items(), key=lambda item: item[1])}
11. **How to handle missing keys?**
12. # Using get()
13. value = my\_dict.get('missing\_key', default\_value)
14. # Using defaultdict
15. from collections import defaultdict
16. dd = defaultdict(int) # default value 0 for missing keys
17. **How to invert a dictionary (swap keys and values)?**
18. original = {'a': 1, 'b': 2, 'c': 3}
19. inverted = {v: k for k, v in original.items()}
20. **How to find the most common elements?**
21. from collections import Counter
22. words = ['apple', 'banana', 'apple',

'cherry', 'banana', 'apple']  
word\_counts = Counter(words)  
print(word\_counts.most\_common(1)) # [('apple', 3)]

### \*\*Dictionary Performance Considerations\*\*:

- Average time complexity for lookup, insert, delete: \*\*O(1)\*\*

- Keys must be hashable (immutable types like strings, numbers, tuples)

- Dictionaries consume more memory than lists for the same number of elements

---

## 7. \*\*Practice Exercises\*\* <a name="exercises"></a>

1. \*\*Word Frequency Counter\*\*: Write a function that takes a string and returns a dictionary with words as keys and their counts as values.

2. \*\*Dictionary Merger\*\*: Write a function that merges two dictionaries. If a key exists in both, the value from the second dictionary should be kept.

3. \*\*Invert Dictionary\*\*: Write a function that inverts a dictionary (keys become values and vice versa). Handle the case where multiple keys have the same value by storing the inverted values in a list.

4. \*\*Nested Dictionary Updater\*\*: Write a function that takes a nested dictionary and a list of keys, and updates the value at the nested key location.

5. \*\*Tuple Key Searcher\*\*: Create a function that searches for a value in a dictionary where keys are tuples, returning all keys that contain the search value in any position of the tuple.

---

### \*\*Solutions\*\*:

```python

# 1. Word Frequency Counter

def word\_frequency(text):

words = text.split()

freq = {}

for word in words:

freq[word] = freq.get(word, 0) + 1

return freq

# 2. Dictionary Merger

def merge\_dicts(dict1, dict2):

return {\*\*dict1, \*\*dict2}

# 3. Invert Dictionary

def invert\_dict(original):

inverted = {}

for key, value in original.items():

if value in inverted:

inverted[value].append(key)

else:

inverted[value] = [key]

return inverted

# 4. Nested Dictionary Updater

def update\_nested(d, keys, value):

for key in keys[:-1]:

d = d.setdefault(key, {})

d[keys[-1]] = value

# 5. Tuple Key Searcher

def search\_tuple\_keys(d, search\_value):

return [key for key in d.keys() if search\_value in key]