Python Dictionary - Complete Notes

What is a Dictionary?

A dictionary in Python is an **unordered collection** of key-value pairs. It's a mutable data structure that stores data in the format `{key: value}`.

Key Characteristics:

- **Mutable**: Can be modified after creation
- **Unordered**: No guaranteed order (Python 3.7+ maintains insertion order)
- **Keys must be unique**: No duplicate keys allowed
- **Keys must be immutable**: strings, numbers, tuples (but not lists)
- **Values can be any data type**: including other dictionaries

---

Creating Dictionaries

```python
# Empty dictionary
dict1 = {}
dict2 = dict()

# Dictionary with initial values
student = {
    'name': 'John',
    'age': 25,
    'grade': 'A',
    'subjects': ['Math', 'Physics']
}

# Using dict() constructor
person = dict(name='Alice', age=30, city='New York')

# From list of tuples
data = dict([('a', 1), ('b', 2), ('c', 3)])
```

Basic Operations

1. Accessing Elements

```python
student = {'name': 'John', 'age': 25, 'grade': 'A'}

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

# Using get() method (safer)
```

```python
print(student.get('name'))  # Output: John
print(student.get('height', 'Not found'))  # Output: Not found
```

2. Adding/Updating Elements

```python
# Adding new key-value pair
student['height'] = 175

# Updating existing value
student['age'] = 26

# Using update() method
student.update({'weight': 70, 'city': 'Mumbai'})
student.update(country='India')
```

3. Removing Elements

```python
# Using del keyword
del student['grade']

# Using pop() - returns the value
age = student.pop('age')  # Returns 26

# Using pop() with default
height = student.pop('height', 0)  # Returns 0 if key doesn't exist

# Using popitem() - removes and returns last inserted key-value pair
last_item = student.popitem()

# Using clear() - removes all elements
student.clear()
```

Dictionary Methods

1. keys(), values(), items()

```python
data = {'a': 1, 'b': 2, 'c': 3}

# Get all keys
print(data.keys())    # dict_keys(['a', 'b', 'c'])

# Get all values
print(data.values())  # dict_values([1, 2, 3])

# Get all key-value pairs
print(data.items())   # dict_items([('a', 1), ('b', 2), ('c', 3)])
```

2. copy() and deepcopy()

```python
import copy

original = {'a': [1, 2, 3], 'b': 4}

# Shallow copy
shallow = original.copy()
shallow['a'].append(4)  # This affects original too!

# Deep copy
deep = copy.deepcopy(original)
deep['a'].append(5)  # This doesn't affect original
```

3. setdefault()

```python
data = {'a': 1, 'b': 2}

# If key exists, return its value; if not, set and return default
result = data.setdefault('c', 3)  # Sets 'c': 3 and returns 3
result = data.setdefault('a', 10)  # Returns 1 (doesn't change existing)
```

4. fromkeys()

```python
# Create dictionary with same value for multiple keys
keys = ['name', 'age', 'city']
default_dict = dict.fromkeys(keys, 'Unknown')
# Output: {'name': 'Unknown', 'age': 'Unknown', 'city': 'Unknown'}

# Be careful with mutable defaults!
bad_dict = dict.fromkeys(['a', 'b'], [])  # Same list object for all keys!
good_dict = {key: [] for key in ['a', 'b']}  # Different list for each key
```

Advanced Operations

1. Dictionary Comprehension

```python
# Basic comprehension
squares = {x: x**2 for x in range(1, 6)}
# Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# With condition
even_squares = {x: x**2 for x in range(1, 11) if x % 2 == 0}

# From existing dictionary
data = {'a': 1, 'b': 2, 'c': 3}
doubled = {k: v*2 for k, v in data.items()}
```

2. Merging Dictionaries

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}
dict3 = {'a': 10, 'e': 5}  # Note: 'a' conflicts with dict1

# Python 3.9+: Using | operator
merged = dict1 | dict2  # {'a': 1, 'b': 2, 'c': 3, 'd': 4}

# Python 3.5+: Using ** unpacking
merged = {**dict1, **dict2, **dict3}  # dict3 overwrites dict1's 'a'

# Using update()
result = dict1.copy()
result.update(dict2)
result.update(dict3)
```

3. Nested Dictionaries

```
students = {
    'student1': {
        'name': 'John',
        'grades': {'math': 85, 'science': 92}
    },
    'student2': {
        'name': 'Alice',
        'grades': {'math': 78, 'science': 88}
    }
}

# Accessing nested values
john_math = students['student1']['grades']['math']

# Safe access with get()
alice_english = students.get('student2', {}).get('grades', {}).get('english', 'Not fo
```

Looping Through Dictionaries

```
data = {'name': 'John', 'age': 25, 'city': 'Mumbai'}

# Loop through keys
for key in data:
    print(key)

# Loop through values
for value in data.values():
    print(value)
```

```python
# Loop through key-value pairs
for key, value in data.items():
    print(f"{key}: {value}")

# Loop with enumerate
for i, (key, value) in enumerate(data.items()):
    print(f"{i}: {key} = {value}")
```

Dictionary as Counter

```python
from collections import Counter

text = "hello world"
char_count = Counter(text)
print(char_count)  # Counter({'l': 3, 'o': 2, 'h': 1, 'e': 1, ' ': 1, 'w': 1, 'r': 1

# Manual counting
manual_count = {}
for char in text:
    manual_count[char] = manual_count.get(char, 0) + 1
```

Interview Tricky Questions & Answers

1. What happens when you use a list as a dictionary key?

```python
# This will raise TypeError
try:
    d = {[1, 2, 3]: 'value'}
except TypeError as e:
    print(f"Error: {e}")  # unhashable type: 'list'

# Use tuple instead
d = {(1, 2, 3): 'value'}  # This works!
```

2. Dictionary with default values - what's the difference?

```python
from collections import defaultdict

# Regular dictionary
regular_dict = {}
# regular_dict['missing_key']  # This raises KeyError

# Using get() with default
value = regular_dict.get('missing_key', [])
```

```python
# Using defaultdict
default_dict = defaultdict(list)
default_dict['missing_key'].append(1)  # No error, creates empty list automatically
```

3. Memory trap with fromkeys()

```python
# WRONG WAY - All keys share the same list object!
wrong_dict = dict.fromkeys(['a', 'b', 'c'], [])
wrong_dict['a'].append(1)
print(wrong_dict)  # {'a': [1], 'b': [1], 'c': [1]} - All affected!

# CORRECT WAY
correct_dict = {key: [] for key in ['a', 'b', 'c']}
correct_dict['a'].append(1)
print(correct_dict)  # {'a': [1], 'b': [], 'c': []} - Only 'a' affected
```

4. Dictionary ordering - version dependent behavior

```python
# Python < 3.7: Order not guaranteed
# Python >= 3.7: Insertion order maintained
d = {}
d['c'] = 3
d['a'] = 1
d['b'] = 2
print(list(d.keys()))  # Python 3.7+: ['c', 'a', 'b']
```

5. Shallow vs Deep copy gotcha

```python
import copy

original = {'numbers': [1, 2, 3], 'name': 'John'}

# Shallow copy
shallow = original.copy()
shallow['numbers'].append(4)
print(original['numbers'])  # [1, 2, 3, 4] - Original affected!

# Deep copy
original = {'numbers': [1, 2, 3], 'name': 'John'}
deep = copy.deepcopy(original)
deep['numbers'].append(4)
print(original['numbers'])  # [1, 2, 3] - Original not affected
```

6. Dictionary comprehension with duplicate keys

```python
# What happens here?
data = [('a', 1), ('b', 2), ('a', 3)]
result = {k: v for k, v in data}
print(result)  # {'a': 3, 'b': 2} - Last value wins!
```

7. Modifying dictionary while iterating

```python
d = {'a': 1, 'b': 2, 'c': 3}

# WRONG - RuntimeError: dictionary changed size during iteration
# for key in d:
#     if d[key] % 2 == 0:
#         del d[key]

# CORRECT - Use list() to create a copy of keys
for key in list(d.keys()):
    if d[key] % 2 == 0:
        del d[key]
```

8. Dictionary equality

```python
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 2, 'a': 1}
print(dict1 == dict2)  # True - Order doesn't matter for equality

# But be careful with nested structures
dict3 = {'list': [1, 2]}
dict4 = {'list': [1, 2]}
print(dict3 == dict4)  # True - Values are compared
print(dict3['list'] is dict4['list'])  # False - Different objects
```

9. What's the output?

```python
d = {'a': 1}
print(d.get('a', []).append(2))  # Output: None (append returns None)
print(d)  # Output: {'a': 1} (original dict unchanged)

# vs
d = {'a': [1]}
d.get('a', []).append(2)
print(d)  # Output: {'a': [1, 2]} (original list modified)
```

10. Dictionary with boolean keys

```python
d = {True: 'yes', False: 'no', 1: 'one', 0: 'zero'}
print(len(d))  # Output: 2 (not 4!)
print(d)       # {True: 'one', False: 'zero'}
# Reason: True == 1 and False == 0 in Python
```

Best Practices

1. **Use get() for safe access**: `dict.get(key, default)` instead of `dict[key]`
2. **Use dict comprehensions**: More readable than loops

3. **Be careful with mutable default values**: Use `defaultdict` or check existence
4. **Use meaningful key names**: Makes code self-documenting
5. **Consider using namedtuple or dataclasses**: For structured data with fixed fields
6. **Use items() for key-value iteration**: More efficient than separate key/value access

---

Time Complexity

Operation	Average Case	Worst Case
Access/Search	O(1)	O(n)
Insert	O(1)	O(n)
Delete	O(1)	O(n)
Copy	O(n)	O(n)

The worst case occurs when there are many hash collisions.