Unit 4 - Lesson 3: Dictionaries Mastery

Quick Reference Guide

🛡️ Safe Access with .get()

The Problem: Accessing a missing key crashes your program!

```
song = {"title": "Blinding Lights", "artist": "The Weeknd"}
print(song["genre"])  # 💥 KeyError: 'genre'
```

The Solution: Use `.get()` for safe access

```
# Returns None if key missing
genre = song.get("genre")        # None

# Returns default if key missing
genre = song.get("genre", "Unknown")  # "Unknown"

# Returns value if key exists (default ignored)
title = song.get("title", "No title")  # "Blinding Lights"
```

When to use each:

Syntax	Use When
`dict["key"]`	Key MUST exist (required field)
`dict.get("key")`	Key MIGHT be missing (optional field)
`dict.get("key", default)`	Need a fallback value

🔧 Dictionary Methods

```
song = {"title": "Blinding Lights", "artist": "The Weeknd", "plays": 3500000}
```

Method	Returns	Example
`.keys()`	All keys	`dict_keys(['title', 'artist', 'plays'])`
`.values()`	All values	`dict_values(['Blinding Lights', 'The Weeknd', 3500000])`
`.items()`	(key, value) pairs	`dict_items([('title', 'Blinding Lights'), ...])`

Useful patterns:

```
# Sum all values
stats = {"views": 1000, "likes": 150, "shares": 25}
total = sum(stats.values())  # 1175

# Check if key exists
if "genre" in song:
    print(song["genre"])
```

🔚 Looping Through Dictionaries

Loop through keys (default):

```
for key in song:
    print(key)  # title, artist, plays
```

Loop through values:

```
for value in song.values():
    print(value)  # Blinding Lights, The Weeknd, 3500000
```

Loop through items (most useful!):

```
for key, value in song.items():
    print(f"{key}: {value}")
```

Output:

```
title: Blinding Lights
artist: The Weeknd
plays: 3500000
```

➕ Updating Dictionaries

Add or update single key:

```
song["genre"] = "Synth-pop"  # Add new key
song["plays"] = 4000000      # Update existing key
```

Update multiple keys at once:

```
song.update({"genre": "Synth-pop", "year": 2020})
```

⚠️ `.update()` modifies the original dictionary!

✨ Dictionary Comprehensions

Basic syntax:

```
{key_expression: value_expression for item in iterable}
```

Transform values:

```
menu = {"latte": 4.50, "mocha": 5.00, "espresso": 3.00}

# Add 8% tax to all prices
with_tax = {item: price * 1.08 for item, price in menu.items()}
# {'latte': 4.86, 'mocha': 5.4, 'espresso': 3.24}
```

Filter with comprehension:

```
scores = {"Alice": 95, "Bob": 67, "Charlie": 82}

# Keep only passing scores
passing = {name: score for name, score in scores.items() if score >= 70}
# {'Alice': 95, 'Charlie': 82}
```

Compare to list comprehensions:

List	Dict
`[expr for item in list]`	`{key: value for item in iterable}`
`[expr for item in list if cond]`	`{key: value for k, v in dict.items() if cond}`

⚠️ Modify vs Return New

Operation	Modifies Original?	Returns
`dict.get(key)`	❌ No	Value or None
`dict.get(key, default)`	❌ No	Value or default
`dict.keys()`	❌ No	Keys
`dict.values()`	❌ No	Values
`dict.items()`	❌ No	(key, value) pairs
`dict.update(other)`	✅ **Yes**	None
`dict["key"] = value`	✅ **Yes**	None
`del dict["key"]`	✅ **Yes**	None

Operation	Modifies Original?	Returns
`{...}` comprehension	❌ No	**New** dict

🎯 Key Takeaways

1. **Use `.get()` for safe access** - never crashes on missing keys
2. **Use `.items()` for looping** - gives both key and value
3. **Dict comprehensions follow list patterns** - `if` at END filters
4. **Know what modifies vs returns** - `.update()` modifies, comprehensions create new

---

📝 Common Patterns

Safe access with default:

```
username = user.get("username", "Anonymous")
```

Loop and filter:

```
for name, score in scores.items():
    if score >= 70:
        print(f"{name} passed!")
```

Transform all values:

```
doubled = {k: v * 2 for k, v in prices.items()}
```

Filter dictionary:

```
filtered = {k: v for k, v in data.items() if v > 0}
```

Check before access:

```
if "email" in user:
    send_email(user["email"])
```