Adding, Removing & Iterating Over Dictionaries

AiCE Software Engineering Foundations — Python Programming

A dictionary is only useful if you can change what it holds and inspect what's inside it. This lesson covers adding and updating key-value pairs, the three ways to remove them, checking membership safely, and the three built-in ways to loop over a dictionary's contents.

1. Adding and Updating Key-Value Pairs

Adding a new key-value pair uses the same syntax as updating an existing one — you assign a value to a key. If the key exists, it's updated. If it doesn't, it's created:

```
developer = {
    "name"    : "Kwame Asante",
    "country" : "Ghana",
}

print(f"Before: {developer}")

# Adding new keys
developer["city"]         = "Accra"
developer["years_coding"] = 3

# Updating an existing key
developer["years_coding"] = 4    # Update from 3 to 4

print(f"After : {developer}")
```

A common pattern for displaying a dictionary cleanly loops through it and formats each field:

```
print("=" * 40)
print("   DEVELOPER PROFILE")
print("=" * 40)
for key, value in developer.items():
    print(f" {key:<15}: {value}")
```

.items() gives you both the key and the value in each iteration — covered fully later in this lesson.

2. Removing Key-Value Pairs

Method	Removes By	Returns the Value?
.pop(key)	Key	Yes
del dict[key]	Key	No
.clear()	Everything — empties the dictionary	No

```
# .pop() – remove by key and return the value
profile = {"name": "Fatima", "age": 24, "temp_token":
"abc123"}
token = profile.pop("temp_token")
print(f"Token removed: {token}")
print(f"Profile      : {profile}")

# del – remove by key
profile2 = {"name": "Emeka", "draft_note": "to be reviewed"}
del profile2["draft_note"]
print(profile2)

# .clear() – empty the entire dictionary
session = {"user": "amara", "token": "xyz789"}
session.clear()
print(session)    # {}
```

3. Checking Keys and Membership

The in keyword checks whether a key exists in a dictionary before you try to access it:

```
student = {
    "name"    : "Aisha Kamara",
    "country" : "Sierra Leone",
    "gpa"     : 3.1,
}

print("name" in student)      # True
print("email" in student)     # False

# Safe pattern: check before accessing
if "gpa" in student:
    print(f"GPA: {student["gpa"]}")
else:
    print("GPA not recorded.")
```

> **The Defensive Programming Pattern**
> *Combining in with an if statement before accessing a key is the standard pattern that prevents KeyErrors in production code — always check before you access, whenever a key's presence isn't guaranteed.*

4. Iterating Over a Dictionary

Python provides three methods for iterating over different parts of a dictionary:

Method	Returns	Use When
.keys()	All keys	You only need the key names
.values()	All values	You only need the stored values
.items()	All key-value pairs	You need both the key and its value

```
african_hubs = {
    "Nigeria": "Lagos",
    "Ghana"  : "Accra",
    "Kenya"  : "Nairobi",
}
```

```
# Iterating over keys
for country in african_hubs.keys():
    print(f" {country}")

# Iterating over values
for city in african_hubs.values():
    print(f" {city}")

# Iterating over key-value pairs (most common)
for country, city in african_hubs.items():
    print(f" {country:<12} -> {city}")
```

> **.items() Is the Most Commonly Used**
> *for country, city in dict.items(): unpacks each key-value pair into two separate variables per iteration — giving you everything you need in a single loop. This is why .items() shows up far more often than .keys() or .values() alone in real code.*

Key Takeaways

Assigning to a key adds it if new or updates it if it already exists; .pop() removes and returns a value, del removes without returning anything, and .clear() wipes a dictionary entirely. Checking in before accessing a key is the standard way to avoid a KeyError. Of the three iteration methods, .keys(), .values(), and .items(), the last is used constantly, since it hands you both the key and value together. The next lesson covers nested dictionaries, lists of dictionaries, and the patterns that make dictionaries feel like a real, structured database.