Dictionaries

Key-Value Storage in Python

The Problem with Lists

```python
# Storing student grades with lists
names = ["Alice", "Bob", "Charlie", "Diana"]
scores = [85, 92, 78, 88]

# Finding Bob's score requires searching
bob_index = -1
for i in range(len(names)):
    if names[i] == "Bob":
        bob_index = i
        break

if bob_index != -1:
    print(f"Bob's score: {scores[bob_index]}")
```

```
Bob's score: 92
```

This:

- Is complicated
- Requires $O(n)$ (where n is number of students) search-time

What if we could look up values directly?

Enter: Dictionaries

A dictionary stores **key-value pairs**:

```python
# Creating a dictionary
grades = {"Alice": 85, "Bob": 92, "Charlie": 78, "Diana": 88}

# Direct lookup!
print(f"Bob's score: {grades['Bob']}")

# Much cleaner than parallel lists!
```

```
Bob's score: 92
```

Key points:

- Curly braces `{}` create a dictionary
- Keys and values separated by `:`
- Items separated by commas
- Look up values using keys

Creating Dictionaries

```python
# Empty dictionary
inventory = {}

# Dictionary with initial values
player_stats = {
    "health": 100,
    "mana": 50,
    "level": 1
}

# Keys can be strings or numbers
room_temps = {
    101: 72,
    102: 68,
    103: 70
}

print(player_stats)
print(room_temps)
```

```
{'health': 100, 'mana': 50, 'level': 1}
{101: 72, 102: 68, 103: 70}
```

The Square Bracket Syntax

Lists

```python
# Accessing list items
scores = [85, 92, 78, 88, 95]
print(scores[0])  # 85
print(scores[1])  # 92
print(scores[4])  # 95
```

```
85
92
95
```

Errors

```python
scores = []
scores[1]
```

Raises an `IndexError` because there's no item at index 1.

Dicts

```python
# Accessing dictionary values
grades = {"Alice": 85, "Bob": 92, "Charlie": 78, "Diana": 8
print(grades["Alice"])  # 85
print(grades["Bob"])    # 92
print(grades["Eve"])    # 95
```

```
85
92
95
```

Errors

```python
grades = {}
grades["Alice"]
```

Raises a `KeyError` because "Alice" is not a key in the dictionary.

Square Bracket vs `.get()`

```python
student = {
    "name": "Alice",
    "age": 20,
    "major": "Computer Science",
    "gpa": 3.8
}

# Get values using keys
print(student["name"])
print(student["gpa"])

# Safe access with get()
height = student.get("height")
print(f"Height: {height}")  # None

# Provide default value
height = student.get("height", "Unknown")
print(f"Height: {height}")
```

```
Alice
3.8
Height: None
Height: Unknown
```

Modifying Dictionaries

```python
inventory = {"sword": 1, "potion": 3, "gold": 50}
print(f"Starting inventory: {inventory}")

# Change a value
inventory["gold"] = 75
print(f"After earning gold: {inventory}")

# Add new items
inventory["shield"] = 1
inventory["arrow"] = 20
print(f"After shopping: {inventory}")

# Remove items
del inventory["sword"]
print(f"Sold sword: {inventory}")
```

```
Starting inventory: {'sword': 1, 'potion': 3, 'gold': 50}
After earning gold: {'sword': 1, 'potion': 3, 'gold': 75}
After shopping: {'sword': 1, 'potion': 3, 'gold': 75, 'shield': 1, 'arrow': 20}
Sold sword: {'potion': 3, 'gold': 75, 'shield': 1, 'arrow': 20}
```

Checking if Keys Exist

```python
menu = {
    "burger": 8.99,
    "fries": 3.99,
    "soda": 2.99,
    "salad": 6.99
}

# Check if item exists
item = "pizza"
if item in menu:
    print(f"{item} costs ${menu[item]}")
else:
    print(f"Sorry, we don't have {item}")
```

```
Sorry, we don't have pizza
```

```python
# Safe pattern for counting
order = {}
items = ["burger", "fries", "burger", "soda", "burger"]

for item in items:
    if item in order:
        order[item] = order[item] + 1
    else:
        order[item] = 1

print(f"Order summary: {order}")
```

```
Order summary: {'burger': 3, 'fries': 1, 'soda': 1}
```

Iterating Over Dictionaries

For-loops can iterate over keys, values, or both:

```python
scores = {"Alice": 85, "Bob": 92, "Charlie": 78}

print("Students:")
for student in scores:
    print(f"  {student}")

print("Grades:")
for student in scores:
    print(f"  {student}: {scores[student]}")
```

We can also get lists:

```python
scores = {"Alice": 85, "Bob": 92, "Charlie": 78}

all_students = list(scores.keys())
all_scores = list(scores.values())
print(f"\nAll students: {all_students}")
print(f"All scores: {all_scores}")
```

```
All students: ['Alice', 'Bob', 'Charlie']

All scores: [85, 92, 78]
```

Iterating with `enumerate()` and `.items()`

Enumerating Lists

We can iterate over lists with both index and value by using `enumerate()`:

```python
fruits = ["apple", "banana", "cherry"]

for i, fruit in enumerate(fruits):
    print(f"{i}: {fruit}")
```

```
0: apple
1: banana
2: cherry
```

Enumerating Dictionaries

Similarly, we can iterate over key-value pairs by using `.items()`:

```python
grades = {"Alice": 85, "Bob": 92, "Charlie": 78}

for student, score in grades.items():
    print(f"{student}: {score}")
```

```
Alice: 85
Bob: 92
Charlie: 78
```

Dictionary Methods

Method	What it does	Example
`get(key)`	Safe lookup	`value = d.get("key")`
`keys()`	Get all keys	`for k in d.keys():`
`values()`	Get all values	`for v in d.values():`
`items()`	Get key-value pairs	`for k,v in d.items():`
`clear()`	Remove all items	`d.clear()`
`pop(key)`	Remove & return	`val = d.pop("key")`

Building Dictionaries Dynamically

```python
# Count letter frequency
text = "hello world"
letter_count = {}

for letter in text:
    if letter != " ":  # Skip spaces
        if letter in letter_count:
            letter_count[letter] = letter_count[letter] + 1
        else:
            letter_count[letter] = 1

print("Letter frequencies:")
for letter, count in letter_count.items():
    print(f"{letter}: {count}")
```

```
Letter frequencies:
h: 1
e: 1
l: 3
o: 2
w: 1
r: 1
d: 1
```

Dictionaries vs Lists

Lists	**Dictionaries**
Ordered by index (0, 1, 2...)	Unordered key-value pairs
Access by position: `lst[0]`	Access by key: `dict["name"]`
Good for sequences	Good for lookups
Can have duplicates	Keys must be unique
`append()` to add	Just assign: `d[key] = value`

Common Dictionary Patterns

Counting

```python
words = ["apple", "banana", "apple", "cherry", "banana" "a
word_count = {}
for word in words:
    if word in word_count:
        word_count[word] = word_count[word] + 1
    else:
        word_count[word] = 1
print(f"Word counts: {word_count}")
```

```
Word counts: {'apple': 3, 'banana': 2, 'cherry': 1}
```

Grouping

```python
grades = [["Alice", 85], ["Bob", 92], ["Alice", 90], ["Bob"
student_grades = {}
for entry in grades:
    name = entry[0]
    grade = entry[1]
    if name not in student_grades:
        student_grades[name] = []
    student_grades[name].append(grade)
print(f"Grouped grades: {student_grades}")
```

```
Grouped grades: {'Alice': [85, 90], 'Bob': [92, 88]}
```

Nested Data Structures

Dictionaries can contain other dictionaries (and lists):

```python
students = {
    "alice": {
        "age": 20,
        "grades": [85, 92, 88],
        "major": "CS"
    },
    "bob": {
        "age": 21,
        "grades": [78, 85, 90],
        "major": "Math"
    }
}
print(students["alice"]["major"])
print(students["bob"]["grades"][0])
```

```
CS
78
```

Lists of Dictionaries for Tables

id	genre	lang	price	qty
1	adventure	fr	11.90	4
2	fantasy	en	8.49	5
3	romance	en	9.99	2
4	fantasy	en	7.99	3
5	adventure	en	9.99	3
6	romance	fr	5.88	1

We can represent tables as lists of dictionaries:

```
books = [
    {"id": 1, "genre": "adventure", "qty": 4, "price": 11.
    {"id": 2, "genre": "fantasy", "qty": 5, "price": 8.49}
    {"id": 3, "genre": "romance", "qty": 2, "price": 9.99}
    {"id": 4, "genre": "fantasy", "qty": 3, "price": 7.99}
    {"id": 5, "genre": "adventure", "qty": 3, "price": 9.9
    {"id": 6, "genre": "romance", "qty": 1, "price": 5.88}
]
```

Reading CSVs as lists of dictionaries is a common pattern in data science and supported by Python's `csv` module.

```
import csv
with open('books.csv', mode='r') as file:
    books = list(csv.DictReader(file))
```

Exercise: Shopping Cart

bigd103.link/shopping-cart