Dictionaries: Labeled Storage

Python for Absolute Beginners · Lesson 13

What you'll learn

- What a dictionary is, and how it differs from a list
- How to make one with keys and values
- How to read, change, and add entries
- When to choose a dictionary over a list
- How to loop through keys and values

Labels instead of positions

- A list finds things by position: item 0, item 1, item 2
- A dictionary finds things by label
- Each entry has two parts: a key (the label) and a value (what is stored under it)
- Entries sit inside curly brackets, written key: value and separated by commas
- A list answers “what is item number 2?” — a dictionary answers “what is the value labeled name?”

Make a dictionary

```
student = {"name": "Mia", "age": 9, "grade": "4th"}
print(student)

{'name': 'Mia', 'age': 9, 'grade': '4th'}
```

Python shows string keys in single quotes — that is just how it displays them, and it means the same thing.

Reading an entry

```
student = {"name": "Mia", "age": 9, "grade": "4th"}
print(student["name"])
print(student["age"])

Mia
9
```

Square brackets, just like a list — but with the key instead of a position. A missing key gives a KeyError.

Changing and adding entries

- Changing an entry and adding one look exactly alike
- The only difference is whether the key already exists
- Assigning to an existing key replaces it
- Assigning to a new key adds a brand-new entry
- Same syntax, two jobs — Python decides by whether the key is already there

Change and add

```
    student = {"name": "Mia", "age": 9, "grade": "4th"}
student["age"] = 10
print(student)
student["pet"] = "Biscuit"
print(student)

{'name': 'Mia', 'age': 10, 'grade': '4th'}
{'name': 'Mia', 'age': 10, 'grade': '4th', 'pet':
'Biscuit'}
```

The age entry was replaced, because “age” already existed; the pet entry was added, because it did not.

List or dictionary?

- List when order matters and items are alike: a week's temperatures, a shopping list
- Dictionary when each item has a name: a student's details, a product's price
- Storing labeled data in a list means memorizing that position 0 is the name and position 1 is the age
- A dictionary removes that memorizing: the label travels with the value

Looping through a dictionary

- A for loop over a dictionary walks its keys
- If you also want the values, .items() gives you both at once
- for key, value in … unpacks each pair into two loop variables at once
- str(value) converts a number to text, because + cannot join text and a number directly

Loop with .items()

```
student = {"name": "Mia", "age": 10, "grade": "4th"}
for key in student:
    print(key)
for key, value in student.items():
    print(key + ": " + str(value))

name
age
grade
name: Mia
age: 10
grade: 4th
```

The first loop prints keys only; the second prints each key and value on its own line.

Hands-on exercise

- Make a dictionary for yourself with keys “name”, “age”, and “city”, and print the whole dictionary
- Print just the value labeled “city”
- Change your age entry to next year's age and print the dictionary again
- Add a new entry for your favorite color and print the dictionary
- Loop through your dictionary with .items() and print each key and value on its own line

Key takeaways

- A dictionary stores entries as key: value pairs inside curly brackets.
- Read with dict[key]; a missing key gives a KeyError.
- Assigning to an existing key replaces it; assigning to a new key adds it.
- Choose a list when order matters, a dictionary when items have labels.
- .items() loops through key-value pairs together; str() converts a number to text for joining.

Take the quiz

Take the Lesson 13 quiz — 5 quick questions to check what you learned.

Next lesson

Debugging: Reading Error Messages and Fixing Bugs