DICTIONARIES

Dictionaries allow you to establish connections between individual pieces of information, such as a state's abbreviation and its full name. There is no limit to the amount of information you can store in a dictionary.

As with lists, you can work with all the information in a dictionary or with any part of a dictionary. These cards show you how to create and work with dictionaries.

4.1 About Dictionaries

4.2 Dictionary Methods

4.3 Looping Through a Dictionary

4.4 Example Dictionaries

4.5 Nesting: A List of Dictionaries

4.6 Nesting: A List in a Dictionary

ABOUT DICTIONARIES

- What is a dictionary?
- How do you define a dictionary?
- How do you get information out of a dictionary?

A *dictionary* is a set of items; each item consists of a *key* and a *value*. You define a dictionary using braces:

```
>>> capitals = {
...     'AK': 'Juneau',
...     'AL': 'Montgomery',
...     'AZ': 'Phoenix',
... }
```

You access the value associated with a key by stating the dictionary name, followed by the key in brackets:

```
>>> capitals['AK']
'Juneau'
```

Add items to an existing dictionary by placing the new key in brackets and providing the value:

```
>>> capitals['AR'] = 'Little Rock'
```

Remove items from a dictionary using the `del` keyword:

```
>>> del capitals['AR']
```

This removes both the key and its associated value from the dictionary.

DICTIONARY METHODS

- What does the dictionary method `get()` do?
- What does the method `keys()` do?
- What does the method `values()` do?
- What does the method `items()` do?

The `get()` method returns the value associated with a key if that key exists in the dictionary. If the key doesn't exist, `get()` returns `None`, not an error.

You can also pass a default value to be returned if the key doesn't exist (example continues from Card 4.1):

```
>>> capitals.get('AK')
'Juneau'
>>> capitals.get('WY')
>>> capitals.get('WY', 'unknown')
'unknown'
```

The methods `keys()`, `values()`, and `items()` return different aspects of a dictionary that also help you loop through the dictionary:

```
>>> capitals.keys()
dict_keys(['AK', 'AL', 'AZ'])
>>> capitals.values()
dict_values(['Juneau', 'Montgomery', 'Phoenix'])
>>> capitals.items()
dict_items([('AK', 'Juneau'),
('AL', 'Montgomery'), ('AZ', 'Phoenix')])
```

Choose one of these methods based on what you're doing with each item.

LOOPING THROUGH A DICTIONARY

- How do you loop through all the keys in a dictionary?
- How do you loop through all the values in a dictionary?
- How do you loop through all the key-value pairs in a dictionary?

Loop through a dictionary to access just the keys. This is the default behavior when looping through a dictionary (example continues from Card 4.1):

```
for state in capitals:
    print(f"State: {state}")
```

```
State: AK
State: AL
State: AZ
```

You can also access just the values in a dictionary:

```
for capital in capitals.values():
    print(f"Capital: {capital}")
```

```
Capital: Juneau
Capital: Montgomery
Capital: Phoenix
```

To work with both keys and values, use the `items()` method:

```
for state, capital in capitals.items():
    print(f"{capital}, {state}")
```

```
Juneau, AK
Montgomery, AL
Phoenix, AZ
```

EXAMPLE DICTIONARIES

- What does a dictionary of multiple similar objects look like?
- What does a dictionary that represents only one object look like?

One use of a dictionary is to represent a collection of key-value pairs with a consistent structure, such as a glossary:

```
python_terms = {
    'loop': 'repeated action',
    'list': 'ordered collection',
    'dictionary': 'keys and values',
}
```

In this example, every key represents a term, and every value represents a meaning.

Another use is to represent information about one kind of object or topic. This dictionary stores information about a musician:

```
c_berry = {
    'first': 'Chuck',
    'last': 'Berry',
    'birth': '10/18/1926',
    'death': '3/18/2017',
    'style': 'rock',
}
```

NESTING: A LIST OF DICTIONARIES

- How do you store a set of dictionaries in a list?
- How do you work with dictionaries stored in a list?

If a dictionary contains a variety of information about an object, it can be helpful to store a collection of dictionaries in a list:

```
musicians = []

e_fitzgerald = {
    'first': 'Ella'
    --snip--
}
musicians.append(ella)

j_joplin = {
    'first': 'Janis',
    --snip--
}
musicians.append(janis)
--snip--
```

To work with this kind of data, loop through the list, working with each dictionary in turn:

```
for musician in musicians:
    full_name = f"{musician['first']} "
    full_name += f"{musician['last']}"
    print(full_name)
```

```
Ella Fitzgerald
Janis Joplin
```

NESTING: A LIST IN A DICTIONARY

- How do you use a list inside a dictionary?
- How do you work with a list that's stored in a dictionary?

A dictionary key can be associated with a list of values:

```
states_visited = {
    'Eric': ['AK', 'WY', 'WA', 'NH'],
    'Isaac': ['CO', 'NY', 'AK'],
    --snip--
}
```

When you loop through the items, each value will be a list. To do this, nest a `for` loop for the list inside a `for` loop for the dictionary:

```
for name, states in states_visited.items():
    print(f"{name} has visited:")
    for state in states:
        print(f"  {state}")
```

```
Eric has visited:
  AK
  WY
  WA
  NH
Isaac has visited:
  CO
  NY
  AK
```