Chapter 6: Dictionaries

BY DR. SABEEHUDDIN HASAN

Dictionaries

- Dictionaries can store an almost limitless amount of information
- You'll learn how to access the information once it's in a dictionary and how to modify that information
- You shall see how to loop through the data in a dictionary
- Understanding dictionaries allows you to model a variety of real-world objects more accurately
- You'll be able to create a dictionary representing a person and then store as much information such as name, age, location and profession of that person

Dictionaries

- **A Simple Dictionary**
  - Consider this dictionary storing information about alien

```
alien_0 = {'color': 'green', 'points': 5}

print(alien_0['color'])

print(alien_0['points'])
```

  - The dictionary alien_0 stores the alien's color and point value.
  - The last two lines access and display that information, as shown here:

```
green
5
```

Working with Dictionaries

- A *dictionary* in Python is a collection of *key-value pairs*
- Each *key* is connected to a value, and you can use a key to access the value associated with that key
- A key's value can be a number, a string, a list, or even another dictionary
- A dictionary is wrapped in braces {} with a series of key-value pairs inside the braces

  alien_0 = {'color': 'green', 'points': 5}
- A *key-value pair* is a set of values associated with each other
- When you provide a key, Python returns the value associated with that key
- Every key is connected to its value by a colon, and individual key-value pairs are separated by commas

  alien_0 = {'color': 'green'}

Working with Dictionaries

- ***Accessing Values in a Dictionary***
  - To get the value associated with a key, give the name of the dictionary and then place the key inside a set of square brackets

    alien_0 = {'color': 'green'}

    print(alien_0['color'])
  - This returns the value associated with the key 'color' from the dictionary alien_0

    green
  - You can have an unlimited number of key-value pairs in a dictionary

    alien_0 = {'color': 'green', 'points': 5}

    new_points = alien_0['points']

    print(f"You just earned {new_points} points!")
  - Once the dictionary has been defined, we pull the value associated with the key 'points' from the dictionary

    You just earned 5 points!

Working with Dictionaries

- ***Adding New Key-Value Pairs***
  - Dictionaries are dynamic structures, and you can add new key-value pairs to a dictionary at any time
  - Let's add two new pieces of information to the alien_0 dictionary: the alien's x- and y-coordinates

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
alien_0['x_position'] = 0
alien_0['y_position'] = 25
print(alien_0)
```

  - When we print the modified dictionary, we see the two additional key-value pairs

```
{'color': 'green', 'points': 5}
{'color': 'green', 'points': 5, 'x_position': 0, 'y_position': 25}
```

  - Dictionaries retain the order in which they were defined

Working with Dictionaries

- ***Starting with an Empty Dictionary***
  - It's sometimes convenient, or even necessary, to start with an empty dictionary and then add each new item to it
  - To start filling an empty dictionary, define a dictionary with an empty set of braces and then add each key-value pair on its own line

```
alien_0 = {}
alien_0['color'] = 'green'
alien_0['points'] = 5
print(alien_0)
```

  - The result is the dictionary we've been using in previous examples

```
{'color': 'green', 'points': 5}
```

Working with Dictionaries

- ***Modifying Values in a Dictionary***
  - To modify a value in a dictionary, give the name of the dictionary with the key in square brackets and then the new value you want associated with that key

```
alien_0 = {'color': 'green'}

print(f"The alien is {alien_0['color']}.")

alien_0['color'] = 'yellow'

print(f"The alien is now {alien_0['color']}.")
```

  - The output shows that the alien has indeed changed from green to yellow:

```
The alien is green.

The alien is now yellow.
```

Working with Dictionaries

- See the following example

```
alien_0 = {'x_position': 0, 'y_position': 25, 'speed': 'medium'}
print(f"Original position: {alien_0['x_position']}")
# Move the alien to the right. Determine how far to move the alien based on its current speed.
if alien_0['speed'] == 'slow':
        x_increment = 1
elif alien_0['speed'] == 'medium':
        x_increment = 2
else:
        x_increment = 3
# The new position is the old position plus the increment.
alien_0['x_position'] = alien_0['x_position'] + x_increment
print(f"New position: {alien_0['x_position']}")
```

Working with Dictionaries

- Because this is a medium-speed alien, its position shifts two units to the right

  Original x-position: 0

  New x-position: 2

- This technique is pretty cool: by changing one value in the alien's dictionary, you can change the overall behavior of the alien
- For example, to turn this medium-speed alien into a fast alien, you would add this line:

  alien_0['speed'] = 'fast'

Working with Dictionaries

- ***Removing Key-Value Pairs***
  - When you no longer need a piece of information that's stored in a dictionary, you can use the *del* statement to completely remove a key-value pair

    alien_0 = {'color': 'green', 'points': 5}

    print(alien_0)

    del alien_0['points']

    print(alien_0)

  - The *del* statement tells Python to delete the key 'points' from the dictionary alien_0 and to remove the value associated with that key as well

    {'color': 'green', 'points': 5}

    {'color': 'green'}

Working with Dictionaries

- ***A Dictionary of Similar Objects***
  - You can also use a dictionary to store one kind of information about many objects
  - For example, you want to poll a number of people and ask them what their favorite programming language is

```
favorite_languages = {
        'jen': 'python',
        'sarah': 'c',
        'edward': 'rust',
        'phil': 'python',
        }
```

  - We've broken a larger dictionary into several lines
  - Each key is the name of a person who responded to the poll, and each value is their language choice

Working with Dictionaries

- ***A Dictionary of Similar Objects***
  - When you know you'll need more than one line to define a dictionary, press ENTER after the opening brace
  - Then indent the next line one level (four spaces) and write the first key-value pair, followed by a comma
  - Once you've finished defining the dictionary, add a closing brace on a new line after the last key-value pair

```
language = favorite_languages['sarah'].title()

print(f"Sarah's favorite language is {language}.")
```

  - We use this syntax to pull Sarah's favorite language from the dictionary and assign it to the variable language

```
Sarah's favorite language is C.
```

Working with Dictionaries

- ***Using get() to Access Values***
  - Using keys in square brackets to retrieve the value want from a dictionary can cause one potential problem:
    - if the key you ask for doesn't exist, you'll get an error

    alien_0 = {'color': 'green', 'speed': 'slow'}

    print(alien_0['points'])
  - This causes error

    print(alien_0['points'])

    ~~~~~^^^^^^

    KeyError: 'points'
    ~~~~~

Working with Dictionaries

- ***Using get() to Access Values***
  - The get() method requires a key as a first argument
  - As a second optional argument, you can pass the value to be returned if the key doesn't exist

```
alien_0 = {'color': 'green', 'speed': 'slow'}

point_value = alien_0.get('points', 'No point value assigned.')

print(point_value)
```

  - If the key 'points' exists in the dictionary, you'll get the value, else we get a clean message instead of an error

```
No point value assigned.
```

Looping Through a Dictionary

- ***Looping Through All Key-Value Pairs***
  - The following dictionary would store one person's username, first name, and last name

```
user_0 = {
        'username': 'efermi',
        'first': 'enrico',
        'last': 'fermi',
        }
```

  - if you want to see everything stored in this user's dictionary, you can use a *for* loop

```
for key, value in user_0.items():
        print(f"Key: {key}")
        print(f"Value: {value}")
```

Looping Through a Dictionary

- ***Looping Through All Key-Value Pairs***
  - To write a for loop for a dictionary, you create names for the two variables that will hold the key and value in each key-value pair
  - You can choose any names you want for these two variables

    for k, v in user_0.items()
  - The for loop then assigns each of these pairs to the two variables provided
  - In the preceding example, we use the variables to print each key, followed by the associated value

    Key: username

    Value: efermi

    Key: first

    Value: enrico

    Key: last

    Value: fermi

Looping Through a Dictionary

- ***Looping Through All Key-Value Pairs***
  - If you loop through the favorite_languages dictionary, you get the name of each person in the dictionary and their favorite programming language

```
favorite_languages = {
        'jen': 'python',
        'sarah': 'c',
        'edward': 'rust',
        'phil': 'python',
        }
for name, language in favorite_languages.items():
        print(f"{name.title()}'s favorite language is {language.title()}.")
```

Looping Through a Dictionary

- ***Looping Through All Key-Value Pairs***
  - Now, in just a few lines of code, we can display all of the information from the poll:

    Jen's favorite language is Python.

    Sarah's favorite language is C.

    Edward's favorite language is Rust.

    Phil's favorite language is Python.

  - This type of looping would work just as well if our dictionary stored the results from polling a thousand or even a million people

Looping Through a Dictionary

- ***Looping Through All the Keys in a Dictionary***
  - The *keys()* method is useful when you don't need to work with all of the values in a dictionary

```
favorite_languages = {
        'jen': 'python',
        'sarah': 'c',
        'edward': 'rust',
        'phil': 'python',
        }
  for name in favorite_languages.keys():
        print(name.title())
```

Looping Through a Dictionary

- ***Looping Through All the Keys in a Dictionary***
  - This for loop tells Python to pull all the keys from the dictionary favorite_languages and assign them one at a time to the variable name
  - The output shows the names of everyone who took the poll:

    Jen

    Sarah

    Edward

    Phil
  - Looping through the keys is actually the default behavior when looping through a dictionary, so this code would have exactly the same output if you wrote:

    for name in favorite_languages:
  - rather than:

    for name in favorite_languages.keys():

Looping Through a Dictionary

- ***Looping Through All the Keys in a Dictionary***

```
favorite_languages = {
    --snip--
    }
friends = ['phil', 'sarah']
for name in favorite_languages.keys():
    print(f"Hi {name.title()}.")
    if name in friends:
        language = favorite_languages[name].title()
        print(f"\t{name.title()}, I see you love {language}!")
```

Looping Through a Dictionary

- ***Looping Through All the Keys in a Dictionary***
  - Everyone's name is printed, but our friends receive a special message:

```
Hi Jen.
Hi Sarah.
    Sarah, I see you love C!
Hi Edward.
Hi Phil.
    Phil, I see you love Python!
```

  - You can also use the keys() method to find out if a particular person was polled

```
if 'erin' not in favorite_languages.keys():
    print("Erin, please take our poll!")
```

Erin, please take our poll!

Looping Through a Dictionary

- ***Looping Through a Dictionary's Keys in a Particular Order***
  - Looping through a dictionary returns the items in the same order they were inserted.
  - Sometimes, though, you'll want to loop through a dictionary in a different order.

```
favorite_languages = {
        'jen': 'python',
        'sarah': 'c',
        'edward': 'rust',
        'phil': 'python',
        }
for name in sorted(favorite_languages.keys()):
        print(f"{name.title()}, thank you for taking the poll.")
```

Looping Through a Dictionary

- ***Looping Through a Dictionary's Keys in a Particular Order***
  - The output shows everyone who took the poll, with the names displayed in ascending order:

    - Edward, thank you for taking the poll.
    - Jen, thank you for taking the poll.
    - Phil, thank you for taking the poll.
    - Sarah, thank you for taking the poll.

Looping Through a Dictionary

- ***Looping Through All Values in a Dictionary***
  - If you are primarily interested in the values that a dictionary contains, you can use the values() method to return a sequence of values without any keys

```
favorite_languages = {
        'jen': 'python',
        'sarah': 'c',
        'edward': 'rust',
        'phil': 'python',
        }
print("The following languages have been mentioned:")
for language in favorite_languages.values():
        print(language.title())
```

Looping Through a Dictionary

- ***Looping Through All Values in a Dictionary***
  - The for statement here pulls each value from the dictionary and assigns it to the variable language.
  - When these values are printed, we get a list of all chosen languages:
  - The following languages have been mentioned:

    Python

    C

    Rust

    Python

Looping Through a Dictionary

- ***Looping Through All Values in a Dictionary***
  - This approach pulls all the values from the dictionary without checking for repeats.
  - This might work fine with a small number of values, but in a poll with a large number of respondents, it would result in a very repetitive list.
  - To see each language chosen without repetition, we can use a set.
  - A *set* is a collection in which each item must be unique:

```
print("The following languages have been mentioned:")
for language in set(favorite_languages.values()):
        print(language.title())
```

Looping Through a Dictionary

- ***Looping Through All Values in a Dictionary***
  - When you wrap set() around a collection of values that contains duplicate items, Python identifies the unique items in the collection and builds a set from those items
  - The result is a nonrepetitive list of languages that have been mentioned by people taking the poll:

    The following languages have been mentioned:

    Python

    C

    Rust
  - *You can build a set directly using braces and separating the elements with commas:*

    >>> ***languages = {'python', 'rust', 'python', 'c'}***

    >>> ***languages***

    {'rust', 'python', 'c'}

Looping Through a Dictionary

- ***Using while Loop to Traverse All Values in a Dictionary***
  - For using a while loop, you must first convert the dictionary to a list

```
key = list (favorite_languages)
print(key)
i = 0
while (i < len(key)):
    print(key[i],':',favorite_languages[key[i]])
    i += 1
```

Looping Through a Dictionary

- ***Using while Loop to Traverse All Values in a Dictionary***
  - The output is as follows

```
['jen', 'sarah', 'edward', 'phil']
jen : python
sarah : c
edward : rust
phil : python
```

Nesting

- *Nesting* happens when you store multiple dictionaries in a list, or a list of items as a value in a dictionary.
- You can nest dictionaries inside a list, a list of items inside a dictionary, or even a dictionary inside another dictionary
- ***A List of Dictionaries***
  - The following code builds a list of three aliens

```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}
alien_2 = {'color': 'red', 'points': 15}
aliens = [alien_0, alien_1, alien_2]
for alien in aliens:
        print(alien)
```

Nesting

- ***A List of Dictionaries***
  - We first create three dictionaries, each representing a different alien.
  - We store each of these dictionaries in a list called aliens 1.
  - Finally, we loop through the list and print out each alien:

    {'color': 'green', 'points': 5}

    {'color': 'yellow', 'points': 10}

    {'color': 'red', 'points': 15}

Nesting

- ***A List of Dictionaries***
  - We use range()to create a fleet of 30 aliens

```
aliens = []
# Make 30 green aliens.
for alien_number in range(30):
        new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
        aliens.append(new_alien)
# Show the first 5 aliens.
for alien in aliens[:5]:
        print(alien)
 print("...")
# Show how many aliens have been created.
print(f"Total number of aliens: {len(aliens)}")
```

Nesting

- ***A List of Dictionaries***

```
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
...
```

  - These aliens all have the same characteristics
  - Python considers each one a separate object, allowing us to modify each alien individually

Nesting

- ***A List of Dictionaries***
  - We can add the following code to change the color, speed and points of the first 3 aliens

```
for alien in aliens[:3]:
    if alien['color'] == 'green':
        alien['color'] = 'yellow'
        alien['speed'] = 'medium'
        alien['points'] = 10
```

Nesting

- ***A List in a Dictionary***
  - Useful to put a list inside a dictionary
  - You might describe a pizza that someone is ordering
  - You could store a list of the pizza's toppings

```
# Store information about a pizza being ordered.
pizza = {'crust': 'thick','toppings': ['mushrooms', 'extra cheese']}
# Summarize the order.
print(f"You ordered a {pizza['crust']}-crust pizza "
"with the following toppings:")
for topping in pizza['toppings']:
    print(f"\t{topping}")
```

Nesting

- ***A List in a Dictionary***
  - The following output summarizes the pizza that we plan to build:

```
You ordered a thick-crust pizza with the following toppings:
    mushrooms
    extra cheese
```

Nesting

- ***A Dictionary in a Dictionary***
  - A dictionary can be nested inside a dictionary

```
users = {
'aeinstein': {'first': 'albert','last': 'einstein','location': 'princeton',},
'mcurie': {'first': 'marie','last': 'curie','location': 'paris',},
}
for username, user_info in users.items():
    print(f"\nUsername: {username}")
    full_name = f"{user_info['first']} {user_info['last']}"
    location = user_info['location']
    print(f"\tFull name: {full_name.title()}")
    print(f"\tLocation: {location.title()}")
```

Nesting

- ***A Dictionary in a Dictionary***
  - The following output is displayed

```
Username: aeinstein
    Full name: Albert Einstein
    Location: Princeton
Username: mcurie
    Full name: Marie Curie
    Location: Paris
```

Nesting

- ***A Dictionary in a Dictionary***

```
# Example shopping cart (items with quantity and price)

cart = {

    "apple": {"quantity": 3, "price": 1.50},

    "banana": {"quantity": 2, "price": 0.75},

    "orange": {"quantity": 4, "price": 1.00}

}
```

Nesting

- ***A Dictionary in a Dictionary***

```
# Calculate the total price of the cart
total_price = 0
for item, details in cart.items():  # Outer loop: iterates through the cart dictionary
    for key, value in details.items():  # Inner loop: iterates through item details
        if key == "quantity":
            quantity = value
        elif key == "price":
            price = value
    total_price += quantity * price  # Multiply quantity and price for each item
print(f"Total Price: ${total_price:.2f}")
```

Nesting

- ***Output:***
  - Total Price: $10.00
- ***Explanation:***
  - The outer loop iterates through each item in the cart (e.g., apple, banana, orange).
  - The inner loop extracts the quantity and price from the dictionary for each item.
  - The total price is calculated by multiplying the quantity by the price for each item.