Chapter 6: Dictionaries

Contents

Red light: No AI

Time required: 90 minutes

DRY

Don't **R**epeat **Y**ourself

Dictionaries

Dictionaries and lists share the following characteristics:

- Both are mutable.
- Both are dynamic. They can grow and shrink as needed.
- Both can be nested. A list can contain another list. A dictionary can contain another dictionary. A dictionary can also contain a list, and vice versa. This is typically how Web API JSON data is presented.

Dictionaries are different from lists in how elements are accessed:

- List elements are accessed by their position in the list, via indexing.
- Dictionary elements are accessed via keys.
- Each key must be unique.

A dictionary can be defined by enclosing a comma-separated list of key-value pairs in curly braces **{}**. A colon **:** separates each key from its associated value:

```
dict = {
    <key>: <value>,
    <key>: <value>,
      .
      .
    <key>: <value>
}
```

The following defines a simple dictionary that maps a location to the name of its corresponding Major League Baseball team.

```
mlb_team = {
    'Colorado' : 'Rockies',
    'Boston'   : 'Red Sox',
    'Minnesota': 'Twins',
    'Milwaukee': 'Brewers',
    'Seattle'  : 'Mariners'
}
```

In common usage, a dictionary is a collection of words matched with their definitions. Given a word, you can look up its definition. Python has a built-in dictionary type called dict which you can use to create dictionaries with arbitrary definitions for character strings. It can be used for the common usage, as in a simple English-Spanish dictionary.

```
"""
   A small English to Spanish dictionary is created,
   The dictionary is accessed.
"""
# Create an empty dictionary
spanish = ()
# Add key value pairs to the dictionary
spanish['hello'] = 'hola'
spanish['yes'] = 'si'
spanish['one'] = 'uno'
spanish['two'] = 'dos'
spanish['three'] = 'tres'
spanish['red'] = 'rojo'
spanish['black'] = 'negro'
spanish['green'] = 'verde'
spanish['blue'] = 'azul'

# Print the entire dictionary
print(spanish)

# Access the value by the key using brackets[]
print(spanish['two'])
print(spanish['red'])

# .get is safer, if the key doesn't exist, there isn't an exception
print(spanish.get("three"))
# The key fray doesn't exist, none is returned, rather than an exception
print(spanish.get("fray"))
```

Example run:

```
{'hello': 'hola', 'yes': 'si', 'one': 'uno', 'two': 'dos', 'three': 'tres', 'red
': 'rojo', 'black': 'negro', 'green': 'verde', 'blue': 'azul'}
dos
rojo
tres
None
```

Explanation

```
# Create an empty dictionary with curly braces
spanish = {}
```

An empty dictionary is created using **{}** curly braces and is assigned the descriptive name **spanish**.

To refer to the definition for a word, you use the dictionary name, follow it by the word inside square brackets. This notation can either be used on the left-hand side of an assignment to make (or remake) a definition, or it can be used in an expression (as in the print functions), where its earlier definition is retrieved. For example,

```
spanish['hello'] = 'hola'
```

makes an entry in our spanish dictionary for 'hello' , with definition 'hola'.

```
print(spanish['red'])
```

retrieves the definition for 'red', which is 'rojo'.

A dictionary is organized by key value pairs.

Here is a list that contains the number of days in the months of the year:

```
days = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

If we want the number of days in January, use **days[0]**. December is **days[11]** or **days[-1]**.

Here is a dictionary of the days in the months of the year:

```
days = {"January":31, "February":28, "March":31, "April":30, "May":31,
       "June":30, "July":31, "August":31, "September":30, "October":31,
       "November":30, "December":31}
```

To get the number of days in January, we use **days["January"]** or **days.get("January")**. One benefit of using dictionaries for this is the code is more readable, and we don't have to figure out which index in the list a given month is at.

Unlike other sequences, items in dictionaries are unordered. The first item in a list named spam would be **spam[0]**. There is no "first" item in a dictionary. The dictionary does remember the order of entry. While the order of items matters for determining whether two lists or tuples are the same, it does not matter in what order the key-value pairs are typed in a dictionary.

Dictionary Basics

Parameter	Details

`key`	The desired key to lookup
`value`	The value to set or return

Literal Syntax

```
d = {} # empty dict
d = {"key": "value"} # dict with initial values
```

Creating Dictionaries

Here is a simple dictionary:

```
d = {"A":100, "B":200}
```

To declare a dictionary, we enclose it in curly braces (`{}`). Each entry consists of a pair separated by a colon. The first part of the pair is called the key and the second is the value. The key acts like an index. In the first pair, "A":100, the key is "A", the value is 100, and d["A"] gives 100.

Keys are often strings. They can be integers, floats, and many other things as well. You can mix different types of keys in the same dictionary and different types of values, too.

Changing Dictionaries

Let's start with this dictionary:

```
d = {"A":100, "B":200}
```

To change **d["A"]** to 400, do this. Dictionaries cannot have duplicate key values, the existing value at the key A is replaced with 400

```
d["A"] = 400
```

To add a new entry to the dictionary, we can just assign it, like below:

```
d["C"] = 500
```

To delete an entry from a dictionary, use the **del** operator:

```
del d["A"]
```

Empty dictionary: The empty dictionary is **{}**, which is the dictionary equivalent of **[]** for lists or **""** for strings.

Tutorial 6.1: Create a Dictionary

You can use a dictionary as an actual dictionary of definitions. Create the following program and save it as **cats_and_dogs.py**

```
"""
    Name: cats_and_dogs.py
    Author:
    Created:
    Purpose: Create and use a dictionary
"""


def main():
    # Define the key : value pairs of the dictionary
    dictionary = {
        'dog': 'has a tail and goes woof!',
        'cat': 'says meow',
        'mouse': 'is chased by cats'
    }

    # Prompt the user to enter a dictionary key
    print(" This dictionary contains values for dog, cat, or mouse.")
    word = input(" Enter a word (key): ")

    # Use the key entered by the user to access the value
    print(f" The value associated with {word}: {dictionary.get(word)}")
    print(f" A {word} {dictionary.get(word)}")


# Call the main function
if __name__ == '__main__':
    main()
```

Example run:

```
This dictionary contains values for dog, cat, or mouse.
Enter a word (key): mouse
The value associated with mouse: is chased by cats
A mouse is chased by cats
```

Dictionary Examples

Example 1 The following dictionary is useful in a program that works with Roman numerals.

```
numerals = {'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}
```

Example 2 In the game Scrabble, each letter has a point value associated with it. We can use the following dictionary for the letter values:

```
points = {'A':1, 'B':3, 'C':3, 'D':2, 'E':1, 'F':4, 'G':2,
          'H':4, 'I':1, 'J':8, 'K':5, 'L':1, 'M':3, 'N':1,
          'O':1, 'P':3, 'Q':10, 'R':1, 'S':1, 'T':1, 'U':1,
          'V':4, 'W':4, 'X':8, 'Y':4, 'Z':10}
```

To score a word, we can do the following dictionary comprehension.

```
score = sum([points[c] for c in word])
```

Or, if you prefer the long way:

```
total = 0
for c in word:
    total += points[c]
```

Example 3 A dictionary provides a nice way to represent a deck of cards:

```
deck = [{'value':i, 'suit':c}
for c in ['spades', 'clubs', 'hearts', 'diamonds']
    for i in range(2, 15)]
```

The deck is a list of 52 dictionaries. The shuffle method can be used to shuffle the deck:

```
shuffle(deck)
```

The first card in the deck is **deck[0]**. To get the value and the suit of the card, we would use the following:

```
deck[0]['value']
deck[0]['suit']
```

Working with Dictionaries

Accessing a Dictionary with [] You can access a dictionary with **[]** by using the key. If the key doesn't exit, it will throw an error

```
dict = {'A':100, 'B':200}
print(dict["A"])
A
```

.get: .get is a safer way to access a dictionary. If the requested key doesn't exist, it doesn't return an error, it returns none.

```
dict = {'A':100, 'B':200}
print(dict.get("A"))
A
```

Copying dictionaries: Just like for lists, making copies of dictionaries is a little tricky. To copy a dictionary, use its **copy** method. Here is an example:

```
d2 = d.copy()
```

in The `in` operator is used to tell if something is a key in the dictionary. For instance, say we have the following dictionary:

```
d = {'A':100, 'B':200}
```

Referring to a key that is not in the dictionary will produce an error. For instance, `print(d['C'])` will fail. To prevent this error, we can use the `in` operator to check first if a key is in the dictionary before trying to use the key. Here is an example:

```
letter = input('Enter a letter: ')
if letter in d:
    print('The value is', d[letter])
else:
    print('Not in dictionary')
```

You can also use **not in** to see if a key is not in the dictionary.

Lists of keys and values: The following table illustrates the ways to get lists of keys and values from a dictionary. It uses the dictionary **d={'A':1,'B':3}**.

Statement	Result	Description
`list(d)`	`['A', 'B']`	keys of d
`list(d.values())`	`[1, 3]`	values of d
`list(d.items())`	`[('A',1),('B',3)]`	(key,value) pairs of d

The pairs returned by `d.items` are tuples.

Here is a use of `d.items` to find all the keys in a dictionary `d` that correspond to a value of 100:

```
d = {'A':100, 'B':200, 'C':100}
L = [x[0] for x in d.items() if x[1]==100]
['A', 'C']
```

dict The **dict** function is another way to create a dictionary. One use for it is kind of like the opposite of the **items** method:

```
d = dict([('A', 100),('B', 300)])
```

This creates the dictionary **{'A':100, 'B':300}**. This way of building a dictionary is useful if your program needs to construct a dictionary while it is running.

Iterating over Dictionaries

Looping: Looping through dictionaries is like looping through lists. Here is an example that prints the keys in a dictionary:

```
for key in d:
    print(key)
```

Here is an example that prints the values:

```
for key in d:
    print(d[key])
```

We will be looping over the keys in the dictionary, not the values.

```
dict = {
    "c": 10,
    "b": 20,
    "a": 30
}
for key in dict:
    print(f"key {key} has value {dict[key]}")
```

This gives us the output:

```
key c has value 10
key b has value 20
key a has value 30
```

In the latest version of Python, the keys come out in the same order they've been added to the dictionary.

If you want them in another order, there are options:

```
dict = {
    "c": 10,
    "b": 20,
    "a": 30
}
for key in sorted(dict): # <--- Add the call to sorted()
    print(f"key {key} has value {dict[key]}")
```

Now we have this, where the keys have been sorted alphabetically.

```
key a has value 10
key b has value 20
key c has value 30
```

Similar to how lists work, you can use the **.items()** method to get all keys and values out at the same time in a for loop, like this:

```
dict = {
    "c": 10,
    "b": 20,
    "a": 30
}
for key, value in dict.items():
    print(f"key {key} has value {value}")
```

Dictionary Functionality

You have several tools in your toolkit for working with dicts in Python:

Method	Description
`.clear()`	Empty a dictionary, removing all keys/values
`.copy()`	Return a copy of a dictionary
`.get(key)`	Get a value from a dictionary, with a default if it doesn't exist
`.items()`	() Return a list-ish of the (key,value) pairs in the dictionary
`.keys()`	Return a list-ish of the keys in the dictionary
`.values()`	Return a list-ish of the values in the dictionary

`.pop(key)`	Return the value for the given key, and remove it from the dictionary
`.popitem()`	pop (remove) and return the most-recently-added (key, value) pair

You can get an item out of a dictionary using the **.get()** method. This method returns a default value of None if the key doesn't exist.

```
val = d.get("x")
if val is None: # Note: use "is", not "==" with "None"!
   print("Key x does not exist")
else:
   print(f"Value of key x is {d[x]}")
```

If you want to see just all the keys or values, you get an iterable back with **.keys()** or **.values()** method.

```
dict = {
    "c": 10,
    "b": 20,
    "a": 30
}
for key in dict.keys():
    print(key) # Prints "c", "b", "a"
for value in dict.values():
    print(value) # Prints 10, 20, 30
```

Tutorial 6.2: Address Book

A dictionary is like an address-book where you can find the address or contact details of a person by knowing only his/her name i.e. we associate keys (name) with values (details). Note that the key must be unique just like you cannot find out the correct information if you have two persons with the exact same name.

You can use only immutable objects (like strings) for the keys of a dictionary. You can use either immutable or mutable objects for the values of the dictionary. You should use only simple objects for keys.

Pairs of keys and values are specified in a dictionary by using the notation:

```
d = {key1: value1, key2: value2}
```

Notice that the key-value pairs are separated by a colon and the pairs are separated themselves by commas and all this is enclosed in a pair of curly braces.

Remember that key-value pairs in a dictionary are not ordered in any manner. If you want a particular order, then you will have to sort them yourself before using it.

The dictionaries that you will be using are instances/objects of the **dict** class.

```python
"""
    Name: address_book_dictionary.py
    Author:
    Created:
    Purpose: Create and work with a dictionary address book
"""


def main():

    # Define the address book
    address_book = {
        "Fred": "fred@hotmail.com",
        "Larry": "larry@wall.org",
        "Mike": "mike@mike.org",
        "Spammer": "spammer@hotmail.com"
    }

    # Print out the address book dictionary
    print_address_book(address_book)

    # Print a specific address by key
    print(f"\nDisplay a specific address")
    print(f"Fred's address is {address_book.get('Fred')}")

    # Print the number of contacts using the len() method
    print(f"\nThere are {len(address_book)} contacts in the address-book")

    # Delete a key-value pair
    print(f"Delete Spammer")
    del address_book["Spammer"]

    # Print the number of contacts
    print(f"\nThere are {len(address_book)} contacts in the address-book")

    # Print out the address book dictionary
    print_address_book(address_book)

    # Add a key-value pair to the dictionary
    print(f"\nAdd Guido to the address book")
    address_book["Guido"] = "guido@pythonstuff.org"

    # Is Guido (in) the address book?
    print(f"\nIs Guido in the address book?")
    if "Guido" in address_book:
        print(f"Guido's address is {address_book.get('Guido')}")

    # Print the address book dictionary
    print("\nThe raw dictionary")
    print(address_book)

    # Call a function to print the address book one entry at a time
    print_address_book(address_book)
```

```python
def print_address_book(address_book):
    """
        Print the address book one entry at a time
    """
    print(f"\nDisplay the address book.")

    # Loop through the dictionary one entry at a time
    # using the .items() method
    # Unpack the dictionary keys and items into two variables
    # Print the variables
    for name, address in address_book.items():
        print(f"Contact {name} at {address}")


# Call the main function
if __name__ == "__main__":
    main()
```

Example run:

```
Display the address book.
Contact Fred at fred@hotmail.com
Contact Larry at larry@wall.org
Contact Mike at mike@mike.org
Contact Spammer at spammer@hotmail.com

Display a specific address
Fred's address is fred@hotmail.com

There are 4 contacts in the address-book
Delete Spammer

There are 3 contacts in the address-book

Display the address book.
Contact Fred at fred@hotmail.com
Contact Larry at larry@wall.org
Contact Mike at mike@mike.org

Add Guido to the address book

Is Guido in the address book?
Guido's address is guido@pythonstuff.org

The raw dictionary
{'Fred': 'fred@hotmail.com', 'Larry': 'larry@wall.org', 'Mike': 'mike@mike.org'
 'Guido': 'guido@pythonstuff.org'}

Display the address book.
Contact Fred at fred@hotmail.com
Contact Larry at larry@wall.org
Contact Mike at mike@mike.org
Contact Guido at guido@pythonstuff.org
```

How It Works

```
# Define the address book
address_book = {
    "Fred": "fred@hotmail.com",
    "Larry": "larry@wall.org",
    "Mike": "mike@mike.org",
    "Spammer": "spammer@hotmail.com"
}
```

We create the dictionary **address_book** using the notation already discussed. We then access key-value pairs by specifying the key using the indexing operator as discussed in the context of lists and tuples. Observe the simple syntax.

```
# Deleting a key-value pair
print(f"Delete Spammer")
del address_book["Spammer"]
```

We can delete key-value pairs using the **del** statement. We specify the dictionary and the indexing operator for the key to be removed and pass it to the **del** statement. There is no need to know the value corresponding to the key for this operation.

```
# Adding a key-value pair
print(f"\nAdd Guido to the address book.")
address_book["Guido"] = "guido@pythonstuff.org"
```

We can add new key-value pairs by simply using the indexing operator to access a key and assign that value, as we have done for Guido in the above case.

```
# Is Guido (in) the address book?
print(f"\nIs Guido in the address book?")
if "Guido" in address_book:
    print(f"Guido's address is {address_book.get('Guido')}")
```

We check if a key-value pair exists using the **in** operator as shown above.

```
def print_address_book(address_book):
    """
        Print the address book one entry at a time
    """
    print(f"\nDisplay the address book.")

    # Loop through the dictionary one entry at a time
    # using the .items() method
    # Unpack the dictionary keys and items into two variables
    # Print the variables
    for name, address in address_book.items():
        print(f"Contact {name} at {address}")
```

We access each key-value pair of the dictionary using the **items()** method of the dictionary which returns a list of tuples where each tuple contains a pair of items - the key followed by the value. We retrieve this pair and assign it to the variables name and address correspondingly for each pair using the for in loop and then print these values in the for block.

Tutorial 6-3: Random Quotes Web API

Web API's return information in JSON (Javascript Object Notation) format. JSON is very similar to a Python dictionary.

Click on the following URL to see the raw JSON format in a web browser.

https://api.quotable.io/random

To access information on the web with Python, we need to install or update a Python module, requests.

```
# Install requests
pip install requests
# Update requests
pip install requests -U
```

This is the specific URL we are using to make our API request. This will return random quote. You can copy and paste this into your code.

```
URL = "https://api.quotable.io/random"
```

Create a Python program named: **random_quotes.py**

Enter the following code.

```
"""
    Name: random_quotes.py
    Author: William A Loring
    Created: 05/27/21
    Purpose: Get random quotes from api.quotable.io
"""

# Import the requests module
import requests
import os

# Set this to False to only display the joke
# this will bypass the display of the conversion process
IS_DEBUGGING = False

# URL for single random jokes
URL = "https://api.quotable.io/random"


def main():
    get_data()

    # Menu loop
    while True:
        # Display menu choices
        answer = input(
            " \nAnother quote?\n [Y] or [Enter] to quit): ")
        # Exit program if Enter is pressed
        # The Enter key returns a null string
        if answer == "":
            break
        # Clear the screen
        os.system('cls' if os.name == 'nt' else 'clear')
        get_data()
```

```
def get_data():
    """Get a random quote from the quotable.io api."""
    # Use the requests.get() function to get a response object
    response = requests.get(URL)
    # If the status_code is 200, successful connection and data
    if (response.status_code == 200):
        # Convert the JSON data into a Python dictionary with key value pairs
        quote_data = response.json()

        # Used to debug process
        if (IS_DEBUGGING == True):

            # Display the status code
            print('\nThe status code for this API request is',
                  response.status_code, "\n")

            # Display the raw JSON data
            print('The raw data from the Quotable API:')
            print(response.text)

            # Display the Python dictionary
            print('\nThe JSON data converted to a Python dictionary:')
            print(quote_data)

        # Print the data using the dictionary created from the API JSON data
        print()
        print(f' {quote_data.get("content")}')
        print(f' --{quote_data.get("author")}')

    else:
        print('API unavailable')


# If a standalone program, call the main function
# Else, use as a module
if __name__ == '__main__':
    main()
```

Example run with IS_DEBUGGING = True:

```
The status code for this API request is 200

The raw data from the Quotable API:
{"_id":"KaIFTRaBv59T","content":"Always seek out the seed of triumph in every ad
versity.","author":"Og Mandino","tags":["famous-quotes","wisdom"],"authorSlug":"
og-mandino","length":55,"dateAdded":"2019-06-13","dateModified":"2019-06-13"}

The JSON data converted to a Python dictionary:
{'_id': 'KaIFTRaBv59T', 'content': 'Always seek out the seed of triumph in every
 adversity.', 'author': 'Og Mandino', 'tags': ['famous-quotes', 'wisdom'], 'auth
orSlug': 'og-mandino', 'length': 55, 'dateAdded': '2019-06-13', 'dateModified':
'2019-06-13'}

 Always seek out the seed of triumph in every adversity.
 --Og Mandino

Another quote?
 [Y] or [Enter] to quit):
```

JSON is in plain text format, very similar to a Python dictionary.

```
1 {'_id': 'yvGg7FErS-y',
2  'content': 'Without courage, wisdom bears no fruit.',
3  'author': 'Baltasar Gracián',
4     'tags': ['famous-quotes', 'wisdom'],
5  'authorSlug': 'baltasar-gracian',
6  'length': 39,
7  'dateAdded': '2020-01-12',
8  'dateModified': '2020-01-12'
9  }
```

Most API JSON data is a combination of lists and dictionaries.

Assignment 1: Language Translator App

Write a Language Translator App that translates English to Spanish.

1. Create a Python program named **english_to_spanish_dictionary.py**
2. Use a main() function.
3. Import and use the **utils.py** module to print a nice title block for your program.
4. Create a dictionary with the following key value pairs.
    a. one uno
    b. two dos
    c. three tres

5. Print out the keys with a loop.
6. Ask the user to input a word to translate.
7. Pass the dictionary to a function.
8. Display the translation. Example run:

```
+--------------------------------+
|  English to Spanish Dictionay  |
+--------------------------------+
These are the Spanish words that you can translate.
one
two
three
Enter the word you wish to translate: three
three translates to: tres
```

Challenges

1. Add more words to your dictionary.
2. Translate to another or multiple languages.
3. Whatever you think might make your translator app more interesting.

Glossary

aliasing A circumstance where two or more variables refer to the same object.

delimiter A character or string used to indicate where a string should be split.

element One of the values in a list (or other sequence); also called items.

equivalent Having the same value.

index An integer value that indicates an element in a list.

identical Being the same object (which implies equivalence).

list A sequence of values.

list traversal Sequential accessing of each element in a list.

nested list A list that is an element of another list.

object Something a variable can refer to. An object has a type and a value.

reference The association between a variable and its value.

Assignment Submission

1. Attach the pseudocode or create a TODO.
2. Attach all tutorials and assignments.
3. Attach screenshots showing the successful operation of each tutorial program.
4. Submit in Blackboard.