

Intermediate Python

Unit 2: Dictionaries

What are dictionaries?

Dictionaries are containers.

Dictionaries are containers

This means that we can store other values in dictionaries,
similar to a `list` .

Let's see some code (list)

```
driver_scores = [  
    ["Max Verstappen", 454],  
    ["Charles Leclerc", 308],  
    ["Sergio Perez", 305],  
    ["George Russell", 275],  
    ["Carlos Sainz", 246]  
]
```

Let's see some code (dictionary)

```
driver_scores = {  
    "Max Verstappen": 454,  
    "Charles Leclerc": 308,  
    "Sergio Perez": 305,  
    "George Russell": 275,  
    "Carlos Sainz": 246  
}
```

Syntax

```
{  
    key1: value1,  
    key2: value2,  
    key3: value3,  
    ...  
}
```

Storing values in dictionaries

Unlike lists, which only contain values, dictionaries are made up of **keys** and **value**.

Keys and values

These keys and values are referred to as **key / value pairs**, and they are what make dictionaries special.

Key / value pairs

Every **value** in a dictionary is associated with a **key**.

Key / value pairs

We use the **key** to access the **value** in the dictionary.

Key / value pairs

As long as we know the **key**, we can access the **value** in the dictionary.

Dictionaries are containers

Again, dictionaries are containers and are similar to lists.

Dictionaries are similar to lists

Many of the things we can do with a list, we can do with a dictionary as well. And vice versa.

Why have dictionaries then?

Certain tasks are better suited for dictionaries.

Certain tasks are better suited for a dictionary

This means that some tasks are faster to accomplish with a dictionary for the programmer (you).

Certain tasks are better suited for a dictionary

Some tasks are also able to be more efficiently performed by Python when using a dictionary.

What dictionaries are good for

Dictionaries shine when it comes time to looking up (getting) items they contain.

Looking up items in a list

This can be done using the `.get` method.

Looking up items in a list

The `.get` method accepts a **key** and returns the corresponding **value**.

Looking up items in a dictionary, an example

```
driver_scores = {  
    "Max Verstappen": 454,  
    "Charles Leclerc": 308,  
    "Sergio Perez": 305,  
    "George Russell": 275,  
    "Carlos Sainz": 246  
}  
  
print(driver_scores.get("Sergio Perez"))
```

Let's compare this to looking up an item in a list

```
driver_scores = [  
    ["Max Verstappen", 454],  
    ["Charles Leclerc", 308],  
    ["Sergio Perez", 305],  
    ["George Russell", 275],  
    ["Carlos Sainz", 246]  
]  
  
for driver_info in driver_scores:  
    if driver_info[0] == "Sergio Perez":  
        print(driver_info[1])
```

How can we work with dictionaries?

Dictionary operations

- Adding and updating an item
- Deleting an item
- Looking up an item, an alternative
- Checking if dictionary contains key

Adding an item

```
driver_scores["Lewis Hamilton"] = 240
```


Deleting an item

```
driver_scores.pop("Max Verstappen")
```

Looking up an item, an alternative

Alternatively, we can look up an item in a dictionary using the `[]` operator, like we do with lists, but we pass it the key instead.

Looking up an item, an alternative

```
print(driver_scores["Max Verstappen"])
```

What happens when we try to look up a key that is not contained in the dictionary?

Looking up missing keys, an example

```
driver_scores = {  
    "Max Verstappen": 454,  
    "Charles Leclerc": 308,  
    "Sergio Perez": 305,  
}  
  
driver_scores.get("Kevin Magnussen")  
# Returns None (nothing)  
  
driver_scores["Kevin Magnussen"]  
# Traceback (most recent call last):  
#   File "<stdin>", line 1, in <module>  
# KeyError: 'Kevin Magnussen'
```

Checking if dictionary contains key

```
"Kevin Magnussen" in driver_scores # returns True or False
```

Why checking if dictionary contains key is important (1)

```
driver_scores = {  
    "Max Verstappen": 454,  
    "Charles Leclerc": 308,  
    "Sergio Perez": 305,  
}  
  
if "Kevin Magnussen" in driver_scores:  
    print(driver_scores.get("Kevin Magnussen"))  
else:  
    print("Unable to find driver's score")
```

Why checking if dictionary contains key is important (2)

```
driver_scores = {  
    "Max Verstappen": 454,  
    "Charles Leclerc": 308,  
    "Sergio Perez": 305,  
}  
  
driver_score = driver_scores.get("Kevin Magnussen")  
  
if driver_score is not None:  
    print(driver_score)  
else:  
    print("Unable to find driver's information")
```


Dictionaries do more!

There are more operations you can perform on dictionaries, such as adding multiple items at once, getting a list with just the values, or a list with just the keys, and more.

Where can we learn more?

Check out the official Python documentation for dictionaries:

<https://docs.python.org/3/library/stdtypes.html#typesmapping>

You can also read the documentation by running this code

```
help(dict)
```

A note on keys and values

Keys are "hashed", this means they are converted by Python behind the scenes into something that can be used as a key.

A note on keys and values

If something is not hashable, then it cannot be used as a **key** in a dictionary.

A note on keys and values

Values do not need to be hashable, meaning anything can be stored as a **value** in a dictionary.

Getting the hash of a value

We can use the `hash` function to get the hash of a value.

Dictionaries and loops

We can loop over every key / value in our dictionaries using `for` loops.

Dictionaries and loops

```
driver_scores = {  
    "Max Verstappen": 454,  
    "Charles Leclerc": 308,  
    "Sergio Perez": 305,  
}  
  
for key in driver_scores:  
    print(key + " score: " + str(driver_scores[key]))
```

Dictionaries and loops

```
driver_scores = {  
    "Max Verstappen": 454,  
    "Charles Leclerc": 308,  
    "Sergio Perez": 305,  
}  
  
for key, value in driver_scores.items():  
    print(key + " score: " + str(value))
```

Messaging application example

Let's say we're building a messaging application that lets us message our friends and we need to store messages that are sent using our application.

Specifically, we want to store...

- the message's sender,
- the recipient,
- the date/time it was sent,
- and the text content.

We can do that in the following way

- **Sender:** Brady
- **Recipient:** Nephi
- **Datetime:** 05/20/2023 14:09:43
- **Content:** Hey Nephi, it's Marcos' birthday tomorrow, what should we get him?

Now in Python

```
message = {  
    "sender": "Brady",  
    "recipient": "Nephi",  
    "datetime": "05/20/2023 14:09:43",  
    "content": "Hey Nephi, it's Marcos' birthday tomorrow, what should we get him?"  
}
```

More messages this time

```
messages = [  
    {  
        "sender": "Brady",  
        "recipient": "Nephi",  
        "datetime": "05/20/2023 14:09:43",  
        "content": "Hey Nephi, it's Marcos' birthday tomorrow, what should we get him?"  
    },  
    {  
        "sender": "Nephi",  
        "recipient": "Brady",  
        "datetime": "05/20/2023 14:15:31",  
        "content": "I've been thinking about this all year long, I have just the thing."  
    },  
    {  
        "sender": "Brady",  
        "recipient": "Nephi",  
        "datetime": "05/20/2023 14:17:21",  
        "content": "Is it the passion fruit cake from Gourmandise?"  
    },  
    {  
        "sender": "Nephi",  
        "recipient": "Brady",  
        "datetime": "05/20/2023 14:15:31",  
        "content": "You bet it is."  
    }  
]
```

Same thing, different name

Dictionaries go by different names in other languages, such as "associative array" and "map".

