Dictionaries

Reference:

- https://docs.python.org/3/library/stdtypes.html#dict
- https://docs.python.org/3/library/stdtypes.html#dictionary-view-objects
- https://docs.python.org/3/tutorial/datastructures.html#dictionaries
- https://docs.python.org/3/tutorial/datastructures.html#looping-techniques

Many programming languages provide an "associative array" datatype which represents an object with named attributes. Associative arrays are said to have "key/value" pairs, where the "key" represents the name of the attribute and the "value" represents the attribute's value.

Python's implementation of the associative array concept is known as a "dictionary". A Python dictionary comprises curly braces (`{}`) containing one or more key/value pairs, with each key separated from its value by a colon (`:`) and each key/value pair separated by a comma (`,`).

Example dictionaries:

```
{}

{"a": 1, "b": 2, "c": 3}

{"a": 1, "b": 2, "c": 3, "fruits": ["apple", "banana", "pear"]} #
dictionaries can contain lists, or even other nested dictionaries

{"first_name": "Ophelia", "last_name": "Clark", "message": "Hello Again"}
```

Each dictionary is similar to a row in a CSV-formatted spreadsheet or a record in a database, where the dictionary's "keys" represent the column names and its "values" represent the cell values.

city	name	league
New York	Yankees	major
New York	Mets	major
Boston	Red Sox	major
New Haven	Ravens	minor

```
[
    {"city": "New York", "name": "Yankees", "league":"major"},
    {"city": "New York", "name": "Mets", "league":"major"},
```

```
    {"city": "Boston", "name": "Red Sox", "league":"major"},
    {"city": "New Haven", "name": "Ravens", "league":"minor"}
]
```

Operations

Access individual object attributes by their key:

```
person = {
    "first_name": "Ophelia",
    "last_name": "Clarke",
    "message": "Hi, thanks for the ice cream!",
    "fav_flavors": ["Vanilla Bean", "Mocha", "Strawberry"]
}

person["first_name"] #> "Ophelia"
person["last_name"] #> "Clark"
person["message"] #> "Hi, thanks for the ice cream!"
person["fav_flavors"] #> ["Vanilla Bean", "Mocha", "Strawberry"]
person["fav_flavors"][1] #> "Mocha" (an array is still an array, even if it
exists inside a dictionary!)
```

Add or update or remove attributes from an object:

```
person = {
    "first_name": "Ophelia",
    "last_name": "Clarke",
    "message": "Hi, thanks for the ice cream!",
    "fav_flavors": ["Vanilla Bean", "Mocha", "Strawberry"]
}

person["message"] = "New Message" # this is mutating

person["fav_color"] = "blue" # this is mutating

del person["fav_flavors"] # this is mutating

person #> {'first_name': 'Ophelia', 'last_name': 'Clark', 'message': 'New
Message', 'fav_color': 'blue' }
```

Its possible to separate the dictionaries keys from its values, and also to iterate through each pair:

```
```

```
person = {
    "first_name": "Ophelia",
    "last_name": "Clarke",
    "message": "Hi, thanks for the ice cream!",
    "fav_flavors": ["Vanilla Bean", "Mocha", "Strawberry"]
}

person.keys()
#> dict_keys(['first_name', 'last_name', 'message', 'fav_flavors'])
list(person.keys())
#> ['first_name', 'last_name', 'message', 'fav_flavors']

person.values()
#> dict_values(['Ophelia', 'Clark', 'Hi, thanks for the ice cream!',
["Vanilla Bean", "Mocha", "Strawberry"]])
list(person.values())
#> ['Ophelia', 'Clark', 'Hi, thanks for the ice cream!', ["Vanilla Bean",
"Mocha", "Strawberry"]]

person.items()
#> dict_items([('first_name', 'Ophelia'), ('last_name', 'Clark'), ('message',
'Hi, thanks for the ice cream!'), ('fav_flavors', ["Vanilla Bean", "Mocha",
"Strawberry"])])

for k, v in person.items():
    print("KEY:", k, "... VALUE:", v)

#> KEY: first ... VALUE: Ophelia
#> KEY: last ... VALUE: Clark
#> KEY: message ... VALUE: Hi, thanks for the ice cream!
#> KEY: fav_flavors ... VALUE: ["Vanilla Bean", "Mocha", "Strawberry"]
```