Reference guide: Dictionaries

By now you've encountered dictionaries and are discovering their power and utility as a data structure in Python. You've also learned that dictionaries provide a way to store and retrieve data using key-value pairs. Data professionals use dictionaries for many tasks, so it's important to be familiar with how they work. This reading is a reference guide about dictionaries. It's designed to help you in your Python learning journey.

Create a dictionary

There are two main ways to create dictionaries in Python:

- Braces: `{}`
- The dict function: `dict()`

When instantiating a list using braces, separate each element with a colon. For example, the following code creates a dictionary containing continents as keys and their smallest countries as values:

```
smallest_countries = {'Africa': 'Seychelles',
                      'Asia': 'Maldives',
                      'Europe': 'Vatican City',
                      'Oceania': 'Nauru',
                      'North America': 'St. Kitts and Nevis',
                      'South America': 'Suriname'
                      }
```

To create an empty dictionary, use empty braces or the `dict()` function:

```
empty_dict_1 = {}
empty_dict_2 = dict()
```

The `dict()` function uses a different syntax, where keys are entered as the function's keyword arguments and values are assigned with an equals operator:

```
smallest_countries = dict(africa='Seychelles',
                          asia='Maldives',
                          europe='Vatican City',
                          oceania='Nauru',
                          north_america='St. Kitts and Nevis',
```

```
                        south_america ='Suriname'
)
```

Notice that, because the keywords cannot be entered as strings, they cannot contain whitespaces.

Some important notes about keys and values:

- **Dictionary keys:** Can be of any *immutable* data type, such as strings, numbers, or tuples
- **Dictionary values:** Can be of any data type—mutable or immutable—including other dictionaries or objects
- Each key can only correspond to a single value; so, for example, this will throw an error:

  ```
  invalid_dict = {'numbers': 1, 2, 3}
  ```

  But if you enclose multiple values within another single data structure, you can create a valid dictionary. For example:

  ```
  valid_dict = {'numbers': [1, 2, 3]}
  ```

Work with dictionaries

Access values

To access a specific value in a dictionary, you must refer to its key using brackets:

```
my_dict = {'nums': [1, 2, 3],
           'abc': ['a', 'b', 'c']
           }
print(my_dict['abc'])
```

To access all values in a dictionary, use the `values()` method:

```
my_dict = {'nums': [1, 2, 3],
           'abc': ['a', 'b', 'c']
           }
print(my_dict.values())
```

Assign new keys

Dictionaries are mutable data structures in Python. You can add to and modify existing dictionaries. To add a new key to a dictionary, use brackets:

```
my_dict = {'nums': [1, 2, 3],
           'abc': ['a', 'b', 'c']
           }

# Add a new 'floats' key
my_dict['floats'] = [1.0, 2.0, 3.0]
print(my_dict)
```

Check if a key exists in a dictionary

To check if a key exists in a dictionary, use the `in` keyword:

```
smallest_countries = {'Africa': 'Seychelles',
                      'Asia': 'Maldives',
                      'Europe': 'Vatican City',
                      'Oceania': 'Nauru',
                      'North America': 'St. Kitts and Nevis',
                      'South America': 'Suriname'
                      }

print('Antarctica' in smallest_countries)
print('Asia' not in smallest_countries)
```

Delete a key-value pair

To delete a key-value pair from a dictionary, use the `del` keyword:

```
my_dict = {'nums': [1, 2, 3],
           'abc': ['a', 'b', 'c']
           }
del my_dict['abc']
print(my_dict)
```

Dictionary methods

Dictionaries are a core Python class. As you've learned, classes package data with tools to work with it. Methods are functions that belong to a class. Dictionaries have a number of built-in methods that are very useful. Some of the most commonly used methods include:

`items()`

Return a view of the `(key, value)` pairs of the dictionary:

```
my_dict = {'nums': [1, 2, 3],
           'abc': ['a', 'b', 'c']
           }
print(my_dict.items())
```

`keys()`

Return a view of the dictionary's keys:

```
my_dict = {'nums': [1, 2, 3],
           'abc': ['a', 'b', 'c']
           }
print(my_dict.keys())
```

`values()`

Return a view of the dictionary's values:

```
my_dict = {'nums': [1, 2, 3],
           'abc': ['a', 'b', 'c']
           }
print(my_dict.values())
```

Note that the objects returned by these methods are view objects. They provide a dynamic view of the dictionary's entries, which means that, when the dictionary changes, the view reflects these changes. Dictionary views can be iterated over to yield their respective data. They also support membership tests.

Additional resources

- For more information about dictionaries, refer to the Python dictionary documentation.

- For more dictionary methods, refer to the Python mapping types documentation.
- For more information about view objects, refer to the Python dictionary view objects documentation.