3.2 - Dictionary

Ha Khanh Nguyen (hknguyen)

1. What is a dictionary?

- Dictionary, `dict`, is a flexibly sized collection of **key-value pairs**, where key and value are Python objects.
- It is also known as *hash map* or *associative array*.

```
empty_dict = {}
empty_list = []
empty_tuple = ()
empty_string = ''
```

```
d1  = {'a' : 'some value', 'b' : [1, 2, 3, 4]}
d1
```

```
## {'a': 'some value', 'b': [1, 2, 3, 4]}
```

2. Elements of a Dictionary

2.1 Access, insert or set elements

- You can access, insert, or set elements using the same syntax as for accessing elements of a list or tuple:

```
# access an element
d1['a']
```

```
## 'some value'
```

```
# insert an element
d1[7] = 'an integer'
d1
```

```
## {'a': 'some value', 'b': [1, 2, 3, 4], 7: 'an integer'}
```

```
# set a different value for an exisiting key
d1['a'] = 1
d1
```

```
## {'a': 1, 'b': [1, 2, 3, 4], 7: 'an integer'}
```

- To check if a dict contains a key, we can use the `in` keyword:

```
'a' in d1
```

```
## True
```

2.2 Delete elements

- Use the `del` keyword to delete a key-value pair from a dict:

```
del d1['b']
d1
```

```
## {'a': 1, 7: 'an integer'}
```

2.3 Iterators of dict's keys and values

```
d1.keys()
```

```
## dict_keys(['a', 7])
```

```
d1.values()
```

```
## dict_values([1, 'an integer'])
```

2.4 Merge dictionaries

- You can merge one dict into another using the `update()` method:

```
d1.update({'b': 'here again', 'c': [1, 2, 3], 9: (1, 2)})
d1
```

```
## {'a': 1, 7: 'an integer', 'b': 'here again', 'c': [1, 2, 3], 9: (1, 2)}
```

3. Creating Dictionary From Sequences

- Let's say you are given two lists `a` and `b` and you want to create a dictionary where the key are elements of `a` and the values are elements of `b`.

```
key_list = ['a', 'b', 'c']
value_list = [1, 2, 3]

mapping = {}

for key, value in zip(key_list, value_list):
  mapping[key] = value

mapping
```

```
## {'a': 1, 'b': 2, 'c': 3}
```

- Now, what happens if the two lists have different length?!
    - Well, let's try that out in Jupyter Notebook!

3.1 The `zip()` function

- Note the `zip()` function that we used in the above code segment, this is a very useful function (especially with for loop).
- `zip()` "pairs" up the elements of a number of lists, tuples, or other sequences to create a list of tuples.

```
seq1 = ['foo', 'bar', 'baz']
seq2 = ['one', 'two', 'three']

zipped = zip(seq1, seq2)
zipped
```

```
## <zip object at 0x7ff51cca0ac8>
```

```
list(zipped)
```

```
## [('foo', 'one'), ('bar', 'two'), ('baz', 'three')]
```

- `zip()` can take an arbitrary number of sequences, and the number of elements it produces is determined by the shortest sequence:

```
seq3 = [False, True]
list(zip(seq1, seq2, seq3))
```

```
## [('foo', 'one', False), ('bar', 'two', True)]
```

3.2 Unzip!

Given a "zipped" sequence, the `zip()` function can also be applied in a clevery way to "unzip" the sequence.

```
pitchers = [('Nolan', 'Ryan'), ('Roger', 'Clemens'), ('Schilling', 'Curt')]
first_names, last_names = zip(*pitchers)
```

```
first_names
```

```
## ('Nolan', 'Roger', 'Schilling')
```

```
last_names
```

```
## ('Ryan', 'Clemens', 'Curt')
```

Exercise

Let `s` be a string that contains a simple mathematical expression, e.g.,

```
s = '1.5 + 2.1'
```

```
s = '10.0-1.6'
```

```
s = '3.1*5.8'
```

```
s = '4.7 /7.2'
```

The expression will only have 2 operands and the operator will be one of the following: +, -, * and /.

Write a program that interpret the expression, then evaluate it and store the result in the result variable.

This lecture note is modified from Chapter 3 of Wes McKinney's Python for Data Analysis 2nd Ed (https://www.oreilly.com/library/view/python-for-data/9781491957653/).