CSI 1401 Week 13 Resources

by Allen Yan

Similar to last week's topic on advanced list manipulation, this week CSI 1401 will introduce some advanced dictionary operations in Python. Dictionaries are less popular than lists in introductory level programming because it has a more complex internal structure and therefore discourages beginners who do not fully understand it. However, dictionaries are incredibly powerful in the correct situation. Without getting too deep into the painstaking details about runtime analysis, Python dictionary is implemented as a **hashmap**, which possesses the advantage of searching for an element instantaneously, compared to the list having to traverse through all of the elements.

As a reminder, I will be leading a group tutoring session on CSI 1401 every Wednesday from 7:00 pm to 8:00 pm (central time). The session will be conducted online via Microsoft Meetings, where I will be available to provide interactive help to students. If the above time window does not work for you, or if you need additional help, Baylor's Tutoring Center also provides 1-on-1 online tutoring appointments. For more information on signing up for group tutoring or individual tutoring, please visit https://www.baylor.edu/tutoring.

Week 13 Important Topics

Chapter 8 – Lists and Dictionaries

- Common dictionary operations
- Dictionary methods
- Iterating over a dictionary

1. Common dictionary operations

This section is more or less a review of dictionary basics. Dictionary in Python can be declared in two ways, the first is to wrap curly braces around key-value pairs, and the second is to use the `dict()` function and pass in the key-value pairs as either arguments or a list of tuples. See example below.

```
# method 1
dict1 = {'Biden': 'Democrat', 'Trump': 'Republican'}
# method 2
dict2 = dict(Biden='Democrat', Trump='Republican')
# method 3
dict3 = dict([('Biden', 'Democrat'), ('Trump', 'Republican')])
```

Note that in the second method, the keys are passed in the function without quotes to indicate them as strings, this is not a mistake. The function takes the raw text and converts them into strings internally, so you would still access the element of `dict2` by using syntax like `dict2['Trump']`. In fact, if you add quotes around the keys in method 2, the function will not compile.

Another thing to note is that since dictionary is not a sequential container, there is no reason (and no way) to perform slicing operations on it. It is also for this same reason that dictionaries will not allow duplicate keys for its elements (just like there can be no duplicate indices in lists, it may be easier to just think of keys are indices for dictionaries), if a value is assigned with an existing key, the existing value associated with that key will be overwritten. See list below for common dictionary operations.

Operation	Description	Example code
my_dict[key]	Indexing operation – retrieves the value associated with key.	`jose_grade = my_dict['Jose']`
my_dict[key] = value	Adds an entry if the entry does not exist, else modifies the existing entry.	`my_dict['Jose'] = 'B+'`
del my_dict[key]	Deletes the key from a dict.	`del my_dict['Jose']`
key in my_dict	Tests for existence of key in my_dict.	`if 'Jose' in my_dict: # ...`

Image snipped from course zyBooks

2. Dictionary methods

Python dictionaries have much less functions compared to list, partly because the [] operator conveniently covers many functions. It can be used to access, search for, and modify an element. Due to all the special characteristics of the dictionary, much of the methods are left out from lack of necessity. See below for a table of common dictionary methods provided by the course textbook.

Dictionary method	Description	Code example	Output

my_dict.clear()	Removes all items from the dictionary.	`my_dict = {'Ahmad': 1, 'Jane': 42}`
		`my_dict.clear()`
		`print(my_dict)`
			`{}`

my_dict.get(key, default)	Reads the value of the key from the dictionary. If the key does not exist in the dictionary, then returns default.	`my_dict = {'Ahmad': 1, 'Jane': 42}`
		`print(my_dict.get('Jane', 'N/A'))`
		`print(my_dict.get('Chad', 'N/A'))`
			`42`
			`N/A`

my_dict1.update(my_dict2)	Merges dictionary my_dict1 with another dictionary my_dict2. Existing entries in my_dict1 are overwritten if the same keys exist in my_dict2.	`my_dict = {'Ahmad': 1, 'Jane': 42}`
		`my_dict.update({'John': 50})`
		`print(my_dict)`
			`{'Ahmad': 1, 'Jane': 42, 'John': 50}`

my_dict.pop(key, default)	Removes and returns the key value from the dictionary. If key does not exist, then default is returned.	`my_dict = {'Ahmad': 1, 'Jane': 42}`
		`val = my_dict.pop('Ahmad')`
		`print(my_dict)`
			`{'Jane': 42}`

Image snipped from course zyBooks

3. Iterating over a dictionary

Recall we established earlier that Python dictionary does not have indices, instead, Python hashes the keys of each entry into unique values, which can then be used to identify their paired values in a lookup table. Because of the lack of indices, there is no way to iterate through a dictionary using index-based looping, leaving only the item-based looping.

There are a few ways to achieve item-based looping, however. When using loop variables to access the contents of a dictionary, Python interpreter can only update the loop variable with the keys of the dictionary, and then you have to access the value using the loop variable updated with the key.

You cannot go the easy route and simply add another loop variable to try and get the associated values as well. Instead, do the following:

```
# normal way
num_calories = dict(Coke=90, Coke_zero=0, Pepsi=94)
for soda in num_calories:
    print('{}: {}'.format(soda, num_calories[soda]))  # works

for soda, calories in num_calories:
    print('{}: {}'.format(soda, calories))  # doesn't work,
can't compile
```

Fortunately, Python has a workaround (as usual) that can save you a bit of time. By using the `items()` function, which returns a view object containing (key, value) tuples, you can include a second loop variable that gets updated with the dictionary values. The code looks like this:

```
# items() way
num_calories = dict(Coke=90, Coke_zero=0, Pepsi=94)
for soda, calories in num_calories.items():
    print('{}: {}'.format(soda, calories))
```

If you are only concerned with accessing the keys of the dictionary (normally you would not, but special cases happen), you can use the `keys()` function, which returns a view object containing just the keys. The code looks like this:

```
# print just the keys
num_calories = dict(Coke=90, Coke_zero=0, Pepsi=94)
for soda in num_calories.keys():
    print(soda)
```

Lastly, you can also do things with just the values and ignore the keys by using the `values()` function. Typically, you would do this for operations that involve looking through every item in the dictionary, such as checking whether a value exists in the dictionary, or printing out all the values. The code looks like this:

```
# print just the values
num_calories = dict(Coke=90, Coke_zero=0, Pepsi=94)
for soda in num_calories.values():
    print(soda)
```

Useful Resources

I hope that the (your course) resources I shared with you this semester were helpful and efficient in addressing your academic needs. As we approach the end of the fall semester, I would like to let you know that there will be one more resource for this class next week, which will help you review the material for your final exam. All resources for this semester can be found here: https://www.baylor.edu/support_programs/index.php?id=967950. Also, please do not forget that the week of November 16 - 20 is the last week where group tutoring sessions will take place. If you wish to attend the last session, please make sure you check out the tutoring center website (https://www.baylor.edu/tutoring) and follow the instructions to register for group tutoring.

- Basic Python Tutorial on GeeksforGeeks: https://www.geeksforgeeks.org/python-programming-language/
    - This page provides links to detailed explanations to many entry-level Python concepts along with examples. I encourage taking a look at it if the examples in your textbook were not clear enough.
- Official Python Documentation: https://docs.python.org/3/
    - This may be a bit heavy-handed for a beginner-level programmer since the official Python documentation is very thorough and technical. However, learning how to read official documentations is crucial to becoming a good programmer, because the official documentation contains information on everything you can find about Python elsewhere and more. Therefore, I encourage slowly easing yourself into learning how to read the official documentation.