Python: Dictionary

7th Lecture

1. Introduction

Lists organize their elements by position. This mode of organization is useful when you want to locate the first element, the last element, or visit each element in a sequence. However, in some situations, the position of a datum in a structure is irrelevant; we're interested in its association with some other element in the structure. For example, you might want to look up Ahmed's phone number but don't care where that number is in the phone book. A dictionary organizes information by association, not position. For example, when you use a dictionary to look up the definition of "mammal," you don't start at page 1; instead, you turn directly to the words beginning with "M." Phone books, address books, encyclopedias, and other reference sources also organize information by association. In computer science, data structures organized by association are also called tables or association lists. In Python, a dictionary associates a set of keys with data values. For example, the keys in Webster's **Dictionary** *comprise the set of words, whereas the associated data values are their definitions*. In this section, we examine the use of dictionaries in the data processing.

2. Dictionary Literals

A Python **dictionary** *is written as a sequence of key/value pairs separated by commas*. These pairs are sometimes called **entries**. ***Dictionaries can be created*** using **a pair of curly braces {}**. Each item in the dictionary consists of **a key**, followed by a colon **:**, which is

followed by a **value**. And each item is separated using commas **,**. The syntax of dictionary as follows:

<dictionaryName> = {<key$_1$: value$_1$, key$_2$: value$_2$, ……>}

Let's take an example:

Phone_book = {'Savannah' : '476-3321', 'Nathaniel' : '351-7743'}

Personal_information = {'Name' : 'Molly', 'Age' : 18}

You can even create an empty dictionary—that is, a dictionary that contains no entries.

dict_emp = { } *# this will create an empty dictionary*

The keys in a dictionary can be data of any immutable types, including other data structures, although keys normally are strings or integers. The associated values can be of any types. Although the entries may appear to be ordered in a dictionary, this ordering is not significant, and the programmer should not rely on it.

3. Adding Keys and Modifying Values

You add a new key/value pair to a dictionary by using the subscript operator [ ]. The form of this operation is the following:

<dictionaryNmae>[<key>] = <value>

Example 1:

```
info = { }
info["name"] = "Sandy"
info["occupation"] = 1
print(info)
```

The output
{'name': 'Sandy', 'occupation': 1}

The subscript is also used to replace a value at an existing key, as follows:

<dictionaryNmae>[<key>] = <modifyValue>

Example 2:

```
info = { }
info["name"] = "Sandy"
info["occupation"] = 1
print(info)

info["occupation"] = "manage"
print(info)
```

The output
{'name': 'Sandy', 'occupation': 1}
{'name' : 'Sandy', 'occupation': 'manager'}

4. Removing Keys

To delete an entry from a dictionary as follows:

del <dictionaryNmae>[<key>]

Example 3:

```
info = {'ali' : 17, 'Ahmed' : 18}
info["Sandy"] = 25
info["ahlaim"] = 30
print(info)
del info["ahlaim"]
print(info)
```

The output
{'ali': 17, 'Ahmed': 18, 'Sandy': 25, 'ahlaim': 30}
{'ali': 17, 'Ahmed': 18, 'Sandy': 25}

Another way removes its key using the method **pop**. This method expects a key and an optional default value as arguments. If the key is in the dictionary, it is removed, and its

associated value is returned. Otherwise, the default value is returned. If pop is used with just one argument, and this key is absent from the dictionary, Python raises an error.

dictionaryNmae.pop(<key>, None)

Example 4:

```
info = {'ali' : 17, 'Ahmed' : 18}
info["Sandy"] = 25
info["ahlaim"] = 30
print(info)
print(info.pop('ali',None))
print(info)
print(info.pop('Rosia',None))
print(info)
```

The output

```
{'ali': 17, 'Ahmed': 18, 'Sandy': 25, 'ahlaim': 30}
17
{'Ahmed': 18, 'Sandy': 25, 'ahlaim': 30}
None
{'Ahmed': 18, 'Sandy': 25, 'ahlaim': 30}
```

5. Traversing a Dictionary

When a **for** loop is used with a dictionary, the loop's variable is bound to each key in an unspecified order. The next code segment prints all of the keys and their values in our info dictionary:

Example 5:

```
info = {'ali' : 17, 'Ahmed' : 18}
info["Sandy"] = 25
info["ahlaim"] = 30
print(info)

for key in info:
   print(key, ':', info[key])
```

The output

```
{'ali': 17, 'Ahmed': 18, 'Sandy': 25, 'ahlaim': 30}
ali : 17
Ahmed : 18
Sandy : 25
ahlaim : 30
```

Example 6: write a Python program that call a function named convert. It expects two parameters: a string representing the number to be converted and a table of associations of digits and its convert hex-to-binary:

```
hexToBinaryTable = {'0':'0000', '1':'0001', '2':'0010',
                    '3':'0011', '4':'0100', '5':'0101',
                    '6':'0110', '7':'0111', '8':'1000',
                    '9':'1001', 'A':'1010', 'B':'1011',
                    'C':'1100', 'D':'1101', 'E':'1110',
                    'F':'1111'}

def convert(number, table):
    "Builds and returns the base two representation of number."
     binary = ' '
     for digit in number:
          binary = table[digit] + binary
     return binary

print (convert('35A', hexToBinaryTable))
```

The output
101001010011

6. Dictionary Methods

Table below summarizes the commonly used dictionary operations:

Method	Description	Examples
len(dictionaryName)	Returns the number of elements in the dictionary.	info = {'ali' : 17, 'Ahmed' : 18} print(len(info)) **Output** 2
dictionaryName.get(key)	Returns the value if the key exists or returns the default if the key does not exist. Raises an error if the default is omitted and the key does not exist	info = {'ali' : 17, 'Ahmed' : 18} print(info.get('ali')) **Output** 17
dictionaryName.keys()	Returns a list of the keys.	info = {'ali' : 17, 'Ahmed' : 18} print(info.keys()) **Output** dict_keys(['ali', 'Ahmed'])

dictionaryName.values()	Returns a list of the values.	info = {'ali' : 17, 'Ahmed' : 18} print(info.values()) **Output** dict_values([17, 18])
dictionaryName.items()	Returns a list of tuples containing the keys and values for each entry.	info = {'ali' : 17, 'Ahmed' : 18} print(info.items()) **Output** dict_items([('ali', 17), ('Ahmed', 18)])
dictionaryName.get(key)	The get() method takes maximum of two parameters: ***key*** - key to be searched in the dictionary ***value*** (optional) - Value to be returned if the key is not found. The default value is None.	person = {'name': 'Phill', 'age': 22} print('Name: ', person.get('name')) print('Age: ', person.get('age')) ***# value is not provided*** print('Salary: ', person.get('salary')) ***# value is provided*** print('Salary: ', person.get('salary', 0.0)) **Output** Name: Phill Age: 22 Salary: None Salary: 0.0
dictionaryName =dict(<key=value>) **or** **dictionaryName=dict( )**	Create Dictionary Using keyword arguments only.	numbers = dict(x=5, y=0) print('numbers = ',numbers) empty = dict() print('empty = ',empty) **Output** numbers = {'x': 5, 'y': 0} empty = {}

7. Exercises

According to your understanding from lecture 6 (List methods), update all these methods based on dictionary concepts**.**

<Best Regards>

Dr. Raaid Alubady