Chp04C Dictionaries

References

1. https://www.w3schools.com/python/python_dictionaries.asp w3school Python Tutorial (dictionaries)

2. https://www.tutorialspoint.com/python3/python_dictionary.htm Tutorialspoint (dictionaries)

3. *https://docs.python.org/3/tutorial/index.html Can also be obtained from Python manual*

4. *Learning Python, 5th (mark Lutz Oreilly 2013) – **chapter 4 - Introducing Python Object Types***

5. *Python for Everybody (Free book) – **chapter 9 Dictionaries***

Definition

A dictionary is a collection which is changeable (**mutable**). In Python, dictionaries are enclosed in curly braces {}, and they have **key:value** pair. key:value pairs are separated from each other by using **colon (:)**. In other languages they are called associative arrays.

Example:

```
studentsScores={"Juma":34.56, "George":67.5,
"Charles":89}
```

In the first entry, `"Juma"` is a **key**, and 34.56 is a **value**.

An **empty** dictionary is denoted by {} and can be created using: `dictName={}`

The dict() constructor

```
dict3=dict()   # creates an empty dictionary
```

dict() constructor builds dictionaries directly from sequences of key-value pairs:

```
dict3=dict([ ("Juma",34.56), ("George",67.5),
("Charles",89)])   #list of tuples
dict4=dict(( ["Juma",34.56], ["George",67.5],
["Charles",89])) #tuple of lists
print(f"dict3: {dict3}")
```

Output:

```
dict3: {'Juma': 34.56, 'George': 67.5,
'Charles': 89} # same as dict4
```

Refer to: Chp04Cex01Dict01.

Key-value pairs can be: list of list, list of tuples, tuples of lists, tuples of tuples.

keys and values

- Dictionary **key** can be almost any Python type, but **MUST** be **immutable** data types, such as strings, numbers, etc
- **Values**, on the other hand, can be any arbitrary Python object.
- **Keys** are unique within a dictionary while **values** may not be. More than one entry per key is not allowed. This means no duplicate key is allowed.
- When duplicate keys are encountered during assignment, **the last assignment (value) wins**

```
d={"A":5, "G":56, "H":78, "G":99}
print(d)  #{'A': 5, 'G': 99, 'H': 78}
```

Accessing dictionary items - I

- Items of a dictionary can be accessed by referring to its key name, inside square brackets.

Example:

```
studentsScores={"Juma":34.56, "George":67.5,
"Charles":89}
```

To get the value for George key,use:

```
studentsScores["George"] which gives 67.5
```

Note: Can't use: `studentsScores[index]`

```
#index=0,1,2..
```

to access dictionary item.

get method

Attempting to access a data item with a key, which is not a part of the dictionary results into KeyError message

`help("dict") – help for dictionary`

Alternatively can use the **get()** method to get value

```
print(studentsScores.get("George"))
```

The **get()** method returns **None** if the key accessed does not exist in the dictionary. See example next slide.

```
if studentsScores.get("George")==None:
    print("key does not exist")
else:
    print(f"Value for key George is
{studentsScores.get('George')}")
```

- Can also specify what to return if the key is not found. Example:
  `studentsScores.get("George","Not Found")`

Returns **"Not Found"** if no "George" in dictionary.

```
if studentsScores.get("George", "Not
Found")=="Not Found":
    print("key does not exist")
else:
    print(f"Value for key George is
{studentsScores.get('George')}")
```

- Value of a specific item can be changed by referring to its key name. Example: change the value of **Charles** from 89 to 65.5:

```
studentsScores["Charles"]=65.5
```

Example: chp04Cex02Dict02

keys and values methods

- **keys** method can also be used to return keys of a dictionary.
- **values** method can be used to return values of a dictionary.

```
studentsScores={"Juma":34.56, "George":67.5,
"Charles":89}
print(studentsScores.keys())
print(type(studentsScores.keys()))
print(studentsScores.values())
print(type(studentsScores.values()))
```

chp04Cex03Dict03

Results:

```
dict_keys(['Juma', 'George', 'Charles'])
<class 'dict_keys'>
dict_values([34.56, 67.5, 89])
<class 'dict_values'>
```

Looping through a dictionary

- Can use for loop to loop through the dictionary to get keys, values

```
#loop through a dictionary and display keys
studentsScores = {"Juma": 34.56, "George":
67.5, "Charles": 89}
item = 1
for key in studentsScores.keys():
    print(f"{item:<6}{key:10}")
    item = item + 1
```

```
Results:
Item  Key
1     Juma
2     George
3     Charles
```

chp04Cex04Dict04

Lists of keys and values

- Can create a list of keys and values using list constructor as follows:

chp04Cex05Dict05

```
print(list(studentsScores.keys()))
print(list(studentsScores.values()))
Results:
['Juma', 'George', 'Charles']
[34.56, 67.5, 89]
```

Then can use list methods to manipulate data in the formed lists**. Example**: valuesList above gives a list of scores of students. To find the average score can use:

```
avg=sum(valuesList)/len(valuesList)
```

Note: You can also convert keys and values to tuple using **tuple()** function.

items method - I

- To return dictionary's key-value pairs, use **items** method. The **items** method returns a view object which contains the **key-value** pairs of the dictionary, as tuples in a list.

Example:

```
studentsScores={"Juma":34.56, "George":67.5,
"Charles":89}
print(studentsDict.items())
print(type(studentsDict.items()))
```

Results:

```
dict_items([('Juma', 34.56), ('George',
67.5), ('Charles', 89)])
<class 'dict_items'>
```

items method - II

- Can use list() constructor to change that view to list as shown in chp04Cex06Dict06
- Can even use loop to print the items:

```
studentsScores={"Juma":34.56,
"George":67.5, "Charles":89}
for item in studentsScores.items():
    print(item)
```

Output:

```
('Juma', 34.56)
('George', 67.5)
('Charles', 89)
('Francis', 78.5)
```

Display key and value pair

- To print a **key** and **value** use a for loop as follows:

chp04Cex07Dict07

```
studentsScores={"John":45.5, "Hamis":67.5,
"Deo":89.5}
#iterators: 1st is key, 2nd is value
print("Key Value")
for key,value in studentsScores.items():
    print(f"{key:7} {value:5.2f}")
```

Output:

```
Key     Value
John    45.50
Hamis   67.50
Deo     89.50
```

- Alternatively, can use key to get the value as follows:

```
studentsScores={"John":45.5, "Hamis":67.5,
"Deo":89.5}
for key in studentsScores.keys():
    print(f"{key:7}
          {studentsScores[key]:5.2f}")
```

Results: similar to previous slide. Refer to chp04Cex07Dict07

in operator

- To determine if a specified **key** is present in a dictionary use the **in** keyword:

```
studentsScores={"John":45.5, "Hamis":67.5,
"Deo":89.5}
name=input("Enter name to search: ")
if name in studentsScores.keys():
    # key is in the dictionary statements
else:
    # key is not in the dictionary statements
```

Refer to: chp04Cex08Dict08

len function

- To determine how many items (key-value pairs) a dictionary have, use the len() function.

```
print(f"There are
{len(studentsScores)} key-value pair
in the dictionary")
```

Adding a new item

Adding an item to the dictionary is done by using a new key and assigning a value to it:

```
newKey=….  #Specify new key
newValue=…. #Specify value for new key
studentsScores[newKey]=newValue
```

Refer to: chp04Cex08Dict08

sorted function - I

To display dictionary in sorted order of keys, use **sorted()** function as follows:

```
studentsScores=……..
for k in sorted(studentsScores.keys()):
    print(k, studentsScores[k])
```

sorted(studentsScores.keys() results into a list of keys in ascending order

Can specify parameter reverse=True for **descending** with sorted() function.

```
…sorted(studentsScores.keys(),reverse=True)
#descending
```

Refer example chp04Cex09Dict09

sorted function - II

To display dictionary in sorted order of keys, use **sorted()** function as follows:

```
studentsScores=……..
for k in sorted(studentsScores.keys()):
    print(k, studentsScores[k])
```

sorted(studentsScores.keys**()** results into a list of keys in ascending order

Can specify parameter reverse=True for **descending** with sorted() function.

```
…sorted(studentsScores.keys(),reverse=True)
#descending
```

Refer example chp04Cex09Dict09

sorted function – III

Can also use sorted on items

```
studentsScores=……..
for key,value in sorted(studentsScores.items()):
    print(key, value)
```

Can specify parameter reverse=True for **descending** with sorted() function

Refer example chp04Cex10Dict10

Removing items from Dictionary - I

- There are several methods to remove items from a dictionary.
- The **pop** method returns the **value** of an item popped and removes the item with the specified key name.

Syntax: dictName.pop(keyName,default)

If the key does not exist, it returns `default`. If `default` is not specified it gives KeyError. So better check first if the key exist before pop or use default to check if the key exists.

```
x=studentsScores.pop("Francis")
y=studentsScores.pop("Michael", “Not found")
```

Removing items from Dictionary - I

- The **popitem** method removes the last item. In versions before 3.7, a **random** item is removed instead. **The popped item is returned as a tuple**

```
dict1={"Juma":34.56, "George":67.5}
poppedItem=dict1.popitem()
print(poppedItem) # ('George', 67.5)
```

You need to check that the dictionary is not empty

Also can assign returned key and value to variables

```
key,value = dict1.popitem() #key and value
```

- The **clear** method empties the dictionary:

```
studentsScores.clear()
```

- The **del** statement removes the item with the specified key name. KeyError if does not exist.

```
del studentsScores["Francis"]
```

- The del statement can be used to delete the dictionary completely:

```
del studentsScores
```

Refer: Chp04Cex11Dict11

Other dictionary methods

https://www.w3schools.com/python/python_dictionaries.asp

Chp04Cex12Dict12

- **copy:** The copy() method returns a copy of the specified dictionary.

```
dict1={…….}
dict2=dict1.copy()
```

Alternatively: use **dict** function:

```
dict1={…….}
dict2=dict(dict1)
```

update method

The **update** method inserts the specified items to the dictionary.

The update method modifies the current dictionary. So you might want to create a copy of the dictionary before operating on the dictionary.

Syntax: `dictName.update(iterable)`

Where iterable can be a dictionary or an iterable object with key value pairs that will be inserted to the dictionary.

Example: Chp04Cex13Dict13

Refer next slide

```
studentsScores={"Juma":34.56, "George":67.5}
dict2={"Charles":89,"Fatuma":67,"Kennedy":56.5}
studentsScores.update(dict2)
print(studentsScores)
```

Output:

```
{'Juma': 34.56, 'George': 67.5, 'Charles': 89,
'Fatuma': 67, 'Kennedy': 56.5}
```

Note: `studentsScores = studentsScores + dict2` – gives error message.

Merging dictionaries - I

Dictionaries can be merged into **a new** dictionary using unpacking operator ** (starting with python v3.5). chp04Cex14mergeDict01

```
dict1={"Juma":34.56, "George":67.5}
dict2={"Charles":89, "Fatuma":67}
dict3=dict(**dict1, **dict2) #can be more than 2 dictionaries
#alternatively: dict3={**dict1, **dict2}
print(dict3)
```

Output:

```
{'Juma': 34.56, 'George': 67.5, 'Charles': 89, 'Fatuma': 67}
```

Merging dictionaries - II

Python 3.9+ has introduced the merge operator (**|**) in the dict class. Using the merge operator, dictionaries can be combined in a single line of code.

chp04Cex15megeDict02

```
dict1={"Juma":34.56, "George":67.5}
dict2={"Charles":89, "Fatuma":67}

dict3=dict1 | dict2

print(f"\ndict3=dict1 | dict2\n {dict3}")
print('\nMerging in place') dict1 |=dict2
print(f"\ndict1 |= dict2 \n dict1: {dict1}")
```

zip function - I

https://docs.python.org/3/library/functions.html#zip

Syntax: `zip(iteratables)`

Returns an iterator of **tuples**, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. Example:

```
names = ['Juma', 'George', 'Charles']
marks = [34.56, 67.5, 89]
newList=list(zip(names,marks))
print(newList)
```

Output:

```
[('Juma', 34.56), ('George', 67.5),
('Charles', 89)]
```

zip function - II

Constructing dictionaries from lists

If separate **lists/tuples** for the keys and values exists, they can be combined into a dictionary using the **zip** function and a **dict** constructor: **Example:** chp04Cex16Zip

```
names = ['Juma', 'George', 'Charles']
marks = [34.56, 67.5, 89]
new_dict_from_lists=dict(zip(names,marks))
print("new_dict_from_lists=dict(zip(names,
marks))")
print(f"new_dict={new_dict_from_lists}")
```

```
Result: {'Juma':34.56,'George':67.5,'Charles': 89}
```

Extra - I

- You can use the **==** and **!=** operators to test whether two dictionaries contain the same items. The order of the items in a dictionary.
- You cannot use the comparison operators (**>**, **>=**, **<=**, and **<**) to compare dictionaries because the items are **not ordered**.
- `Importing variables defined in .py file`

```
from filename import var1, var2, …
```

From that point you can access the variable by using its name. Refer to example:chp04Cex17ImportVars

Which type of collection to use?

- Lists and Tuples objects are sequences. A dictionary is a table of key-value pairs. List and tuple is an ordered collection of items. Dictionary is unordered collection.
- Lists and dictionaries objects are mutable i.e. it is possible to add new item or delete an item from it. Tuples objects are immutable. Addition or deletion operations are not possible on tuple object.
- Most of the time **lists** and **dictionaries** are used, because of their mutability: can easily change the values of things if we need to.
- Remember: lists ([]), tuples(()), dictionaries({})

Extra

- As a data scientist, you can study also **sets** to know how they work.
- Sets contains unique values (no duplicates are allowed)
- Sets are **NOT** part of IS671 syllabus

==End of chapter 4C ==

Next: Chp05 User defined Functions