Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #34
Dictionary and it's Operations

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU

Verified by,
PCPS Team - 2023

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M(Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

Dictionary in Python

Introduction

Dictionary is a **Non-primitive Non-Linear (Non-sequence type) Data Structure that organizes the data in the form of key and value pairs separated by a colon(:)** . In short, it is named as dict type. It is an **unordered collection of key and value pairs.** Let us try to understand why this type is required through an example.

The requirement is to store the srn and other details of students like name, address, phno and age. If we store in 5 different lists, we are not building any relation between the data. Also, if we delete one element from ages list or names list, other lists will not know that name and age is not available for this particular student. To store all these details in a related manner, we use dictionary by using **SRN as the key and value being the list containing name, address, phno and age.** Thus, it becomes easy to build relation between all the data using the dictionary.

Characteristics of Dicts:

- Can contain **0 or more elements/items in the form of key : value.**
- **Each key is of any immutable type associated with a single value.**
- **Values can be of any type.**
- Elements or items(key-value pair) in dictionary is unordered type, which means that the order isn't decided by the programmer but rather the interpreter based on the concept of Hashing.
- It does not allow duplicate keys. If you assign a new value to the key which is already there in the dictionary, the previous value will be overwritten.
- Dict is **non indexable** - Elements **cannot be accessed using indexing operation.**
- Dict is **mutable** - Can grow or shrink as its size is not fixed.
- Dict is **iterable –** Can get one element at a time. It is eager and not lazy.
- It has number of **built-in functions to manipulate itself.**

Syntactically, elements of the dictionary must be inside **flower brackets(curly braces) – {**

and } and **all elements must be in the form of a key: value pair separated by a comma** between each item.

FYR: Hashing is a mechanism to convert the given element in to an integer. An object is hashable if it has a hash value which never changes during its lifetime.

- **Hashable objects:** int, float, complex, bool, string, tuple, range, frozenset, bytes, decimal . Immutable
- **Unhashable objects:** list, dict, set, bytearray

A. Creation of dict variables

Example 1:

```
d1 = {"r":"red","g":"gren","b":"blue"}
print(type(d1))   #  <class 'dict'>
```

Example 2:

```
d2 = {"r":"red","b":"brown","g":"gren","b":"blue"}
print(d2)   #  {'r': 'red', 'b': 'blue', 'g': 'gren'}
```

```
>>> d3 = {[1,2,3]:"red","b":"brown","g":"gren"}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> d3 = {"r":"red",{"b","B"}:"brown","g":"gren"}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
>>> d2 = {"r":"red","b":["brown","blue"],"g":"gren"}
>>> d2
{'r': 'red', 'b': ['brown', 'blue'], 'g': 'gren'}
>>> ■
```

Note: Empty dict can be created using {} or dict() # constructor function

Observe that when you print the dict variable, the order is not same as how you have given in the creation of dict variable.

B. Accessing the elements of the dict

The dict variable is not indexable but iterable. To access the value, we have to make use of key.

```
d1 = {"r":"red","g":"gren","b":"blue",1:111,"o":"orange"}
print(d1["r"])  #red
print(d1["B"])  #Keyerror
```

To access each element of the dictionary individually, we can use the for loop**. By default, the dictionary is iterated always using only the keys**. Then some expression using the key helps us to access the value.

```
d1 = {"r":"red","g":"gren","b":"blue",1:111,"o":"orange"}
for k in d1:
    print(k,d1[k])
```

```
C:\Users\Dell>python practice_dict.py
r red
g gren
b blue
1 111
o orange
```

Think about this:

Can you iterate through the dictionary using list() ,tuple() or set() ? If yes, what will be the output?

Can you access the elements of the dictionary using index? In the above example d1, what is d1[1] and d1[0] ?

Common Operations on dict:

- len(): Returns the number of items in a dictionary
- min(): Returns the smallest key in a dictionary
- max(): Returns the largest key in a dictionary.
- sum(): Displays the sum of all the keys from a dictionary.
- …

```
>>> d1 =  {10:101010,2:222,6:666,1:111, 1:1111}
>>> d1
{10: 101010, 2: 222, 6: 666, 1: 1111}
>>> len(d1)
4
>>> min(d1)
1
>>> max(d1)
10
>>> sum(d1)
19
>>> sorted(d1)
[1, 2, 6, 10]
>>>
```

```
>>> d1 =  {10:101010,2:222,"6":666,"one":1, 1:1111}
>>> d1
{10: 101010, 2: 222, '6': 666, 'one': 1, 1: 1111}
>>> min(d1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and 'int'
>>> max(d1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '>' not supported between instances of 'str' and 'int'
>>> len(d1)
5
>>> sum(d1)
Traceback (most recent call last):
  File "<stdin>". line 1. in <module>
```

C. Common operators on dicts

- | operator: This union operator is used to combine two dicts together, results in a dict that contains all items from the given two dicts.
- Membership operators: in, not in – This checks whether the specified key is in dictionary or not.
- Relational operators: == and != - Used to compare two dictionary for it's equality.

```
>>> d1 = {1:111,7:777,3:333}
>>> d2 = {7:777,1:111,3:333}
>>> d1
{1: 111, 7: 777, 3: 333}
>>> d2
{7: 777, 1: 111, 3: 333}
>>> d3 = {6:666,4:444,1:111}
>>> d3
{6: 666, 4: 444, 1: 111}
>>> d1|d3
{1: 111, 7: 777, 3: 333, 6: 666, 4: 444}
>>> d1 | d2
{1: 111, 7: 777, 3: 333}
>>> d1 == d2
True
>>> d1 != d3
True
>>> d1 != d2
False
>>> 111 in d1
False
>>> 1 not in d1
False
>>>
```

Note: Usage of +, - , *, >, <,<=, >=, & and ^ doesn't make any sense on dicts.

D. Modifying the elements of the dict

All specific functions of set can be listed using dir() function.

```
>>> dir(dict)
['__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__di
r__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__ior__', '__i
ter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__reduce__', '_
_reduce_ex__', '__repr__', '__reversed__', '__ror__', '__setattr__', '__setitem__', '_
_sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items'
, 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

keys() – Creates the object providing a view on only keys.

```
>>> d1 = {"c":"cranberry","a":"apple","b":"banana"}
>>> d1
{'c': 'cranberry', 'a': 'apple', 'b': 'banana'}
>>> d1.keys()
dict_keys(['c', 'a', 'b'])
>>> for k in d1.keys():
...     print(k, end = ' ')
...
c a b
>>>
```

values() - Creates the object providing a view on only values.

```
>>> for v in d1.values():
...     print(v)
...
cranberry
apple
banana
>>>
```

items() - Creates the object providing a view on all items in the form of a tuple – (key, value).

```
>>> for k,v in d1.items():      #object containing the tuples. Each tuple is (key,value)
...     print(k,v)      #then unpacking happens. k and v gets the key and value
...
c cranberry
a apple
b banana
>>>
```

update()- Updates the current dictionary by adding items from any other dictionary.

```
>>> d1
{'c': 'cranberry', 'a': 'apple', 'b': 'banana'}
>>> d1.update({"d":"diarymilk"})
>>> d1
{'c': 'cranberry', 'a': 'apple', 'b': 'banana', 'd': 'diarymilk'}
>>> d1.update({"d":"diarymilk","g":"grapes"})
>>> d1
{'c': 'cranberry', 'a': 'apple', 'b': 'banana', 'd': 'diarymilk', 'g': 'grapes'}
```

pop()- Removes the specified key and return the corresponding value. If the key is not found, return the default if given; otherwise, raise a KeyError.

popitem() - Removes and return a (key, value) pair as a 2-tuple. Raises KeyError if the dict is empty.

get()- Return the value for key if key is in the dictionary, else default.

fromkeys()- Creates a new dictionary with keys from iterable and values set to value.

setdefault()-Inserts key with a value of default if key is not in the dictionary. Return the value for key if key is in the dictionary,

```
>>> d1
{'c': 'cranberry', 'a': 'apple', 'b': 'banana'}
>>> d1["a"]
'apple'
>>> d1["A"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'A'
>>> d1.get("a")
'apple'
>>> d1.get("A")
>>> d1.pop("A")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'A'
>>> d1.pop("a")
'apple'
>>> d1
{'c': 'cranberry', 'b': 'banana'}
>>>
```

clear() - Removes all the elements in a dict.

```
>>> d1 = {"c":"cranberry","a":"apple","b":"banana"}
>>> d2 = d1.fromkeys(d1)
>>> d2
{'c': None, 'a': None, 'b': None}
>>> d2["a"] = "aeroplane"
>>> d2
{'c': None, 'a': 'aeroplane', 'b': None}



>>>d1.setdefault("a")
'apple'
>>> d1.setdefault("A","AEROPLANE")
'AEROPLANE'
>>> d1
{'c': 'cranberry', 'a': 'apple', 'b': 'banana', 'A': 'AEROPLANE'}
>>>d1.setdefault("d")
>>> d1
{'c': 'cranberry', 'a': 'apple', 'b': 'banana', 'A': 'AEROPLANE',
'd': None}


>>>d1.clear()
>>> d1
{}
```

Few functions for practice are here: popitem() and copy()

Points to think!

- **Can a dictionary have another dictionary as the key?**
- **Can a dictionary have another dictionary as the value?**
- **Which function helps you to create a dictionary by using all the keys of the existing dictionary?**
- **How do you set the default value for all the keys in a dictionary?**
- **Can you sort the dictionary items based on the value? Try this!**

- END -