Dictionaries

Introduction

- Python provides us various options to store multiple values under one variable name.
- Dictionaries is also a collection like a string, list and tuple.
- It is a very versatile data type.
- There is a key in it with a value.(Key:value)
- Dictionaries are mutable data types and it is an unordered collection in the form of key:value
- In List, index of a value is important whereas in dictionary, key of a value is important.

Dictionary Creation

- To create a dictionary, it is needed to collect pairs of key:value in “{ }”.
  - **<dictionary-name>={ <key1>:<value1>,<key2>:<value2>,<key3>:<value3>. . . }**

Example:

teachers={“Rajeev”:”Math”, “APA”:”Physics”,”APS”:”Chemistry:”SB”:”CS”}

In above given example :

Key-value pair	Key	Value
“Rajeev”:”Math”	“Rajeev”	“Math”
“APA”:”Physics”	“APA”	“Physics”
“APS”:”Chemistry”	“APA”	“Chemistry”
“SB”:”CS”	“SB”	“CS”

Dictionary Creation

- Some examples of Dictionary are-

Dict1= { } # this is an empty dictionary without any element.

DayofMonth= { January”:31, ”February”:28, ”March”:31, ”April”:30, ”May”:31, ”June”:30,
”July”:31, ”August”:31, ”September”:30, ”October”:31, ”November”:30,
”December”:31}

FurnitureCount = { “Table”:10, “Chair”:13, “Desk”:16, “Stool”:15, “Rack”:15 }

– By above examples you can easily understand about the keys and their values.
– One thing to be taken care of is that keys should always be of immutable type.

Note: Dictionary is also known as associative array or mapping or hashes .

Dictionary Creation

- Keys should always be of immutable type.

- If you try to make keys as mutable, python shown error in it. For example-

```
>>> dict = {[2,3]:"MyRoom"}
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    dict = {[2,3]:"MyRoom"}
TypeError: unhashable type: 'list'
```

Here key is a list which is of mutable type.

Here error shows that you are trying to create a key of mutable type which is not permitted.

Accessing a Dictionary

- To access a value from dictionary, we need to use key similarly as we use an index to access a value from a list.
- We get the key from the pair of Key: value.

teachers={“Rajeev”:”Math”, “APA”:”Physics”,”APS”:”Chemistry:”SB”:”CS”}

- If we execute following statement from above example-

```
>>> teachers={"Rajeev":"Math", "APA":"Physics", "APS":"Chemistry", "SB":"CS"}
>>> teachers["Rajeev"]
'Math'
>>> print("Rajeev Teaches ", teachers["Rajeev"])
Rajeev Teaches  Math
```

- We have selected key “Rajeev” and on printing it, Math got printed. Another example-

```
>>> d={"Vowel1":'a', "Vowel2":'e', "Vowel3":'i', "Vowel4":'o', "Vowel5":'u'}
>>> print(d["Vowel2"])
e
>>> print(d["Vowel5"])
u
```

If we access a non-key, error will come.

Traversal of a Dictionary

- To traverse a Dictionary, we use for loop. Syntax is-

for <item> in <dictionary>:

```
>>> d={5:"number","a":"String",(1,2):"tuple"}
>>> for k in d:
        print(k," : ",d[k])
```

```
5  :  number
a  :  String
(1, 2)  :  tuple
```

Here, notable thing is that every key of each pair of dictionary d is coming in k variable of loop. After this we can take output with the given format and with print statement.

Assignment : Develop a dictionary of your friends in which key will be your friend's name and his number will be its value.

Traversal of Dictionary

- To access key and value we need to use keys() and values().for example-

```
>>> d={"Vowel1":'a', "Vowel2":'e', "Vowel3":'i', "Vowel4":'o', "Vowel5":'u'}
>>> d.keys()
dict_keys(['Vowel1', 'Vowel2', 'Vowel3', 'Vowel4', 'Vowel5'])
>>> d.values()
dict_values(['a', 'e', 'i', 'o', 'u'])
```

- d.keys( ) function will display only key.
- d.values ( ) function will display value only.

Features of Dictionary

1. **Unordered set:** dictionary is a unordered collection of key:value pairs.
2. **Not a sequence:** like list, string and tuple , it is not a sequence because it is a collection of unordered elements whereas a sequence is a collection of indexed numbers to keep them in order.
3. Keys are used for its indexing because according to Python key can be of immutable type. String and numbers are of immutable type and therefore can be used as a key. Example of keys are as under-

```
>>> d={0:"Key0",1:"Key1","3":"KeyAsString",(4,5):"KeyAsTuple","Hello":6}
>>> d[0]
'Key0'
>>> d[1]
'Key1'
>>> d["3"]
'KeyAsString'
>>> d[(4,5)]
'KeyAsTuple'
>>> d["Hello"]
6
```

Key of a Dictionary should always be of immutable type like number, string or tuple whereas value of a dictionary can be of any type.

Features of Dictionary

4. **Keys should be unique :** Because keys are used to identify values so they should be unique.
5. Values of two unique keys can be same.
6. Dictionary is mutable hence we can change value of a certain key. For this, syntax is-

<dictionary>[<key>] = <value>

```
>>> d={0:"Key0",1:"Key1","3":"KeyAsString",(4,5):"KeyAsTuple","Hello":6}
>>> d["3"]="This is String"
>>> d
{0: 'Key0', 1: 'Key1', '3': 'This is String', (4, 5): 'KeyAsTuple', 'Hello': 6}
```

4. Internally it is stored as a mapping. Its key:value are connected to each other via an internal function called ***hash-function**.* Such process of linking is knows as mapping.

**Hash-function is an internal algorithm to link a and its value.

Working with Dictionary

- Here we will discuss about various operation of dictionary like element adding, updation, deletion of an element etc. but first we will learn creation of a dictionary.
- **Dictionary initialization-** For this we keep collection of pairs of key:value separated by comma (,) and then place this collection inside "{ }".

```
>>> Employee={'name':'suresh','salary':15000,'age':34}
>>> Employee
{'name': 'suresh', 'salary': 15000, 'age': 34}
```

- **Addition of key:value pair to an empty dictionary.** There are two ways to create an empty dictionary-

1. Employee = { }
2. Employee = dict( )

After that use following syntax-

<dictionary>[<key>] = <value>

```
>>> Employee = {}
>>> Employee['name']='Pankaj'
>>> Employee['salary']=20000
>>> Employee
{'name': 'Pankaj', 'salary': 20000}
```

Working with Dictionary

3. **Creation of a Dictionary with the pair of name and value**: dict( ) constructor is used to create dictionary with the pairs of key and value. There are various methods for this-

I. By passing Key:value pair as an argument:

```
>>> Employee=dict(name='Ramesh',salary=10000,age=24)
>>> Employee
{'name': 'Ramesh', 'salary': 10000, 'age': 24}
```

The point to be noted is that here no inverted commas were placed in argument but they came automatically in dictionary.

II. By specifying Comma-separated key:value pair-

```
>>> Employee=dict({'name':'Rahul','age':24})
>>> Employee
{'name': 'Rahul', 'age': 24}
```

Working with Dictionary

III. By specifying Keys and values separately:

For this, we use zip() function in dict ( ) constructor-

```
>>> Employee=dict(zip(('name','salary','age'),('Mukesh',12000,26)))
>>> Employee
{'name': 'Mukesh', 'salary': 12000, 'age': 26}
```

IV. By giving Key:value pair in the form of separate sequence:

```
>>> Employee=dict([['name','anand'],['salary',14000],['age',23]])
>>> Employee
{'name': 'anand', 'salary': 14000, 'age': 23}
```

By passing List

```
>>> Employee=dict((['name','Angad'],['salary',11000],['age',29]))
>>> Employee
{'name': 'Angad', 'salary': 11000, 'age': 29}
```

By passing tuple of a list

```
>>> Employee=dict((('name','Suman'),('salary',17000),('age',21)))
>>> Employee
{'name': 'Suman', 'salary': 17000, 'age': 21}
```

By passing tuple of tuple

Adding an element in Dictionary

following syntax is used to add an element in Dictionary-

```
>>> Employee=dict((('name','Suman'),('salary',17000),('age',21)))
>>> Employee
{'name': 'Suman', 'salary': 17000, 'age': 21}
>>> Employee['Dept']='Sales'
>>> Employee
{'name': 'Suman', 'salary': 17000, 'age': 21, 'Dept': 'Sales'}
```

Nesting in Dictionary

look at the following example carefully in which element of a dictionary is a dictionary itself.

```
Employee={'mukesh':{'age':23,'salary':34000},'Meena':{'age':27,'salary':24000}}
for key in Employee:
    print("Employee",key,':')
    print('Age: ',str(Employee[key]['age']))
    print('Salary: ',str(Employee[key]['salary']))
```

```
Employee mukesh :
Age:  23
Salary:  34000
Employee Meena :
Age:  27
Salary:  24000
```

Updation in a Dictionary

following syntax is used to update an element in Dictionary-

<dictionary>[<ExistingKey>]=<value>

```
>>> Employee={'name':'Sudha','Salary':10000,'age':24}
>>> Employee['Salary']=15000
>>> Employee
{'name': 'Sudha', 'Salary': 15000, 'age': 24}
```

WAP to create a dictionary containing names of employee as key and their salary as value.

File Edit Format Run Options Window Help

```
n=int(input("Enter the Number of Employees"))
Employee={}   #Empty Dictionary
for a in range(n):
    key=input("Enter Name of the Employee")
    value=int(input("Enter Salary of Employee"))
    Employee[key]=value
print("The Dictionary is now is :")
print(Employee)
```

Output

```
Enter the Number of Employees3
Enter Name of the EmployeePavan
Enter Salary of Employee30000
Enter Name of the EmployeeRakesh
Enter Salary of Employee12000
Enter Name of the EmployeeMukesh
Enter Salary of Employee20000
The Dictionary is now is :
{'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
```

Deletion of an element from a Dictionary

following two syntaxes can be used to delete an element form a Dictionary. For deletion, key should be there otherwise python will give error.

1. del <dictionary>[<key>]- it only deletes the value and does not return deleted value.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> del emp['Pavan']
>>> emp
{'Rakesh': 12000, 'Mukesh': 20000}
```

Value did not return after deletion.

2. <dictionary>.pop(<key>) it returns the deleted value after deletion.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> emp.pop('Rakesh')
12000
>>> emp.pop('Aman',"Not Found")
'Not Found'
```

Value returned after deletion.

If key does not match, given message will be printed.

Detection of an element from a Dictionary

Membership operator is used to detect presence of an element in a Dictionary.

1. <key> in <dictionary> it gives true on finding the key otherwise gives false.
2. <key> not in <dictionary> it gives true on not finding the key otherwise gives false.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> 'Pavan' in emp
True
>>> 'Hari' in emp
False
>>> 'Hari' not in emp
True
>>> 'Pavan' not in emp
False
```

*** in and not in does not apply on values, they can only work with keys.**

Pretty Printing of a Dictionary

To print a Dictionary in a beautify manner, we need to import ***json module.*** After that following syntax of dumps ( ) will be used.

json.dumps(<>,indent=<n>)

```
>>> import json
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> print(json.dumps(emp,indent=2))
{
  "Pavan": 30000,
  "Rakesh": 12000,
  "Mukesh": 20000
}
```

Program to create a dictionary by counting words in a line

```
import json
statement="His Name is Pankaj. \
His father is a teacher. His \
father is a good person"
w=statement.split()
d={}
for c in w:
    key=c
    if key not in d:
        count=w.count(key)
        d[key]=count
print("Counting frequencies in list\n",w)
print(json.dumps(d,indent=1))
```

```
Counting frequencies in list
 ['His', 'Name', 'is', 'Pankaj.', '
His', 'father', 'is', 'a', 'teacher
.', 'His', 'father', 'is', 'a', 'go
od', 'person']
{
 "His": 3,
 "Name": 1,
 "is": 3,
 "Pankaj.": 1,
 "father": 2,
 "a": 2,
 "teacher.": 1,
 "good": 1,
 "person": 1
}
```

Here a dictionary is created of words and their frequency.

Dictionary Function and Method

1. ***len( )*** Method : it tells the length of dictionary.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> len(emp)
3
```

2. ***clear( )*** Method : it empties the dictionary.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> emp.clear()
>>> emp
{}
```

3. ***get( )*** Method : it returns value of the given key.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> emp.get('Rakesh')
12000
>>> emp.get('Ankit',"Not Found")
'Not Found'
```

It works similarly as <dictionary>[<key>]

On non finding of a key, default message can be given.

Dictionary Function and Method

4. ***items( )*** Method : it returns all items of a dictionary in the form of tuple of (key:value).

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> mylist=emp.items()
>>> mylist
dict_items([('Pavan', 30000), ('Rakesh', 12000), ('Mukesh', 20000)])
```

5. ***keys( )*** Method : it returns list of dictionary keys.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> emp.keys()
dict_keys(['Pavan', 'Rakesh', 'Mukesh'])
```

6. ***values( )*** Method : it returns list of dictionary values.

```
>>> emp={'Pavan': 30000, 'Rakesh': 12000, 'Mukesh': 20000}
>>> emp.values()
dict_values([30000, 12000, 20000])
```

Dictionary Function and Method

7. ***Update ( )*** Method: This function merge the pair of key:value of a dictionary into other dictionary. Change and addition in this is possible as per need. Example-

```
>>> emp1={'name':'Suresh','salary':10000,'age':24}
>>> emp2={'name':'siya','salary':45000,'dept':'sales'}
>>> emp1.update(emp2)
>>> emp1
{'name': 'siya', 'salary': 45000, 'age': 24, 'dept': 'sales'}
>>> emp2
{'name': 'siya', 'salary': 45000, 'dept': 'sales'}
```

In the above given example, you can see that change is done in the values of similar keys whereas dissimilar keys got joined with their values.

Thank you