Dictionaries

July 26, 2022

Python Data Structures - Dictionaries

Dictionaries

Dictionaries are Python's implementation of a data structure, generally known as associative arrays, hashes, or hashmaps. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value. Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces.

Characteristics of Dictionaries :

Dictionaries and lists share the following characteristics :

Both are mutable.

Both are dynamic. They can grow and shrink as needed.

Both can be nested. A list can contain another list. A dictionary can contain another dictionary.

Dictionaries differ from lists primarily in how elements are accessed :

List elements are accessed by their position in the list, via indexing.

Dictionary elements are accessed via keys.

Creating a Dictionary

There are following three ways to create a dictionary :

Using curly brackets : Enclosing the comma-separated Key: Value pairs inside the {} curly brackets.

Using dict ( ) constructor : Create a dictionary by passing the comma-separated key: value pairs inside the dict().

Using sequence having each item as a pair (key-value)

[63]:
```
#You can create a dictionary by placing a comma-separated list of key:value␣
 ↪pairs in curly braces {}.
#Each key is separated from its associated value by a colon :

# empty Dict with {}
emptyDict = {}
print("Empty Dictionary : ")
print(emptyDict ,"-->", type(emptyDict))
print()
```

```python
# dictionary with integer keys
integerKeys = {1: 'apple', 2: 'ball'}
print("Dictionary With Integer Keys : ")
print(integerKeys ,"-->", type(integerKeys))
print()

# dictionary with mixed keys and value as a list
mixedKeys = {'name': 'John', 1: [2, 4, 3]}
print("Dictionary With Mixed Keys : ")
print(mixedKeys ,"-->", type(mixedKeys))
print()

# using dict()
emptyDict = dict()
print("Empty Dictionary With Dict Constructor : ")
print(emptyDict ,"-->", type(emptyDict))
print()

# using dict() with string keys (keyword arguments)
myDict = dict(name = 'Bob',
              age = 25,
              job = 'Dev')
print("String Keys Dictionary With Dict Constructor : ")
print(myDict ,"-->", type(myDict))
print()

# create a dictionary from sequence having each item as a pair
person = dict([("name", "Mark"), ("country", "USA"), ("telephone", 1178)])
print("Dictionary From Sequence Having Each Item As A Pair : ")
print(person,"-->", type(person))
```

```
Empty Dictionary :
{} --> <class 'dict'>

Dictionary With Integer Keys :
{1: 'apple', 2: 'ball'} --> <class 'dict'>

Dictionary With Mixed Keys :
{'name': 'John', 1: [2, 4, 3]} --> <class 'dict'>

Empty Dictionary With Dict Constructor :
{} --> <class 'dict'>

String Keys Dictionary With Dict Constructor :
{'name': 'Bob', 'age': 25, 'job': 'Dev'} --> <class 'dict'>

Dictionary From Sequence Having Each Item As A Pair :
```

```
{'name': 'Mark', 'country': 'USA', 'telephone': 1178} --> <class 'dict'>
```

[8]:
```
#Other Ways to Create Dictionaries

# Create a dictionary with list of zipped keys/values
keys = ['name', 'age', 'job']
values = ['Bob', 25, 'Dev']
D = dict(zip(keys, values))
print(D)
print()

# Initialize dictionary with default value '0' for each key
keys = ['a', 'b', 'c']
defaultValue = 0
D = dict.fromkeys(keys,defaultValue)
print(D)
```

```
{'name': 'Bob', 'age': 25, 'job': 'Dev'}

{'a': 0, 'b': 0, 'c': 0}
```

[9]:
```
#Creating a Nested Dictionary
people = {1: {'name': 'John', 'age': '27', 'sex': 'Male'},
          2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}}
print(people)
```

```
{1: {'name': 'John', 'age': '27', 'sex': 'Male'}, 2: {'name': 'Marie', 'age':
'22', 'sex': 'Female'}}
```

[30]:
```
#len of Dict
price = {'milk': 1.02, 'coffee': 2.5, 'bread': 2.5}
# count number of keys present in  a dictionary
print(len(price))
```

```
3
```

Important Properties of a Dictionary Keys

Each key is linked to a specific value. Once stored in a dictionary, you can later obtain the value using just the key. For example, consider the Phone lookup where it is very easy and fast to find the phone number(value) when we know the name(Key) associated with it..

Keys must be unique : A key can appear in a dictionary only once.

Key must be immutable type : such as numbers, strings, booleans or tuples.

A dictionary value can be of any type, and duplicates are allowed in that.

[65]:
```
#Keys Must Be Unquie
#Even if you specify a key more than once during the creation of a dictionary,
#the last value for that key becomes the associated value.
```

```
D = {'name': 'Bob','age': 25,
     'name': 'Jane','age':26}
print(D)
```

```
{'name': 'Jane', 'age': 26}
```

[66]:
```
#Key must be immutable type:
D = {(2,2): 25,
     True: 'a',
     'name': 'Bob'}
print(D)
```

```
{(2, 2): 25, True: 'a', 'name': 'Bob'}
```

[68]:
```
#Key must be immutable type:
D = {{2,2}: 25,
     True: 'a',
     'name': 'Bob'}
print(D)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-68-1f427633522b> in <module>
      1 #Key must be immutable type:
----> 2 D = {{2,2}: 25,
      3      True: 'a',
      4      'name': 'Bob'}
      5 print(D)

TypeError: unhashable type: 'set'
```

[69]:
```
#Value can be of any type
#Values of different datatypes
D = {'a':[1,2,3],
     'b':{1,2,3}}
print(D)

# duplicate values
D = {'a':[1,2],
     'b':[1,2],
     'c':[1,2]}
print(D)
```

```
{'a': [1, 2, 3], 'b': {1, 2, 3}}
{'a': [1, 2], 'b': [1, 2], 'c': [1, 2]}
```

Accessing Elements Of A Dictionary

There are two different ways to access the elements of a dictionary.

Retrieve value using the key name inside the [ ] square brackets

Retrieve value by passing key name as a parameter to the get ( ) method of a dictionary.

If we use the square brackets [ ], KeyError is raised in case a key is not found in the dictionary. On the other hand, the get ( ) method returns None if the key is not found.

[70]:
```
# get vs [] for retrieving elements
my_dict = {'name': 'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))

# Trying to access keys which doesn't exist throws error
# Output None
print(my_dict.get('address'))

# KeyError
print(my_dict['address'])
```

```
Jack
26
None
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-70-2ec6dc01caba> in <module>
     13
     14 # KeyError
---> 15 print(my_dict['address'])

KeyError: 'address'
```

[42]:
```
#Retreving Values from Nested Dict
# each dictionary will store data of a single student
jessa = {'name': 'Jessa', 'state': 'Texas', 'city': 'Houston', 'marks': 75}
emma = {'name': 'Emma', 'state': 'Texas', 'city': 'Dallas', 'marks': 60}
kelly = {'name': 'Kelly', 'state': 'Texas', 'city': 'Austin', 'marks': 85}

# Outer dictionary to store all student dictionaries (nested dictionaries)
class_six = {'student1': jessa, 'student2': emma, 'student3': kelly}

# Get student3's name and mark
```

```
print("Student 3 name:", class_six['student3']['name'])
print("Student 3 marks:", class_six['student3']['marks'])
```

```
Student 3 name: Kelly
Student 3 marks: 85
```

Get all keys and values

Use the following dictionary methods to retrieve all key and values at once :

keys ( ) : Returns the list of all keys present in the dictionary.

values ( ) : Returns the list of all values present in the dictionary

items ( ) : Returns all the items present in the dictionary. Each item will be inside a tuple as a key-value pair.

[24]:
```
person = {"name": "Jessa", "country": "USA", "telephone": 1178}

# Get all keys
print(person.keys(),"-->",type(person.keys()))
print()

# Get all values
print(person.values(),"-->",type(person.values()))
print()

# Get all key-value pair
print(person.items(),"-->",type(person.items()))
```

```
dict_keys(['name', 'country', 'telephone']) --> <class 'dict_keys'>

dict_values(['Jessa', 'USA', 1178]) --> <class 'dict_values'>

dict_items([('name', 'Jessa'), ('country', 'USA'), ('telephone', 1178)]) -->
<class 'dict_items'>
```

Iterate Over A Dictionary In Python

[25]:
```
#Dict of States and Dicts

statesAndCapitals = {
    'Gujarat': 'Gandhinagar',
    'Maharashtra': 'Mumbai',
    'Rajasthan': 'Jaipur',
    'Bihar': 'Patna'
}
```

[26]:
```
#Iterate through all keys in a dictionary

print('List Of given states:\n')
```

```
# Iterating over keys
for state in statesAndCapitals:
    print(state)
```

```
List Of given states:

Gujarat
Maharashtra
Rajasthan
Bihar
```

[27]:
```
#Iterate through all Values in a dictionary

print('List Of given capitals:\n')

# Iterating over values
for capital in statesAndCapitals.values():
    print(capital)
```

```
List Of given capitals:

Gandhinagar
Mumbai
Jaipur
Patna
```

[28]:
```
#Iterate through all key, value pairs in a dictionary

print('List Of given states and their capitals:\n')

# Iterating over key and values
for state, capital in statesAndCapitals.items():
    print(state, ":", capital)
```

```
List Of given states and their capitals:

Gujarat : Gandhinagar
Maharashtra : Mumbai
Rajasthan : Jaipur
Bihar : Patna
```

[29]:
```
#Iterate through all key, value pairs in a dictionary

print('List Of given states and their capitals:\n')

# Iterating over key and values
for i in statesAndCapitals:
    print(i, '->', statesAndCapitals[i])
```

```
List Of given states and their capitals:

Gujarat -> Gandhinagar
Maharashtra -> Mumbai
Rajasthan -> Jaipur
Bihar -> Patna
```

[41]:
```python
#iterate through Nested Dictonary
# address dictionary to store person address
address = {"state": "Texas", 'city': 'Houston'}

# dictionary to store person details with address as a nested dictionary
person = {'name': 'Jessa', 'company': 'Google', 'address': address}

# Display dictionary
print("person:", person)

# Get nested dictionary key 'city'
print("City:", person['address']['city'])
print()
# Iterating outer dictionary
print("Person details")
print()
for key, value in person.items():
    if key == 'address':
        # Iterating through nested dictionary
        print("Person Address")
        for nested_key, nested_value in value.items():
            print(nested_key, ':', nested_value)
    else:
        print(key, ':', value)
```

```
person: {'name': 'Jessa', 'company': 'Google', 'address': {'state': 'Texas',
'city': 'Houston'}}
City: Houston

Person details

name : Jessa
company : Google
Person Address
state : Texas
city : Houston
```

Adding items to the dictionary

We can add new items to the dictionary using the following two ways :

Using key-value assignment : Using a simple assignment statement where value can be assigned directly to the new key.

Using update( ) Method : In this method, the item passed inside the update() method will be inserted into the dictionary

[31]:
```
person = {"name": "Jessa", 'country': "USA", "telephone": 1178}

# update dictionary by adding 2 new keys
person["weight"] = 50
person.update({"height": 6})

# print the updated dictionary
print(person)
```

```
{'name': 'Jessa', 'country': 'USA', 'telephone': 1178, 'weight': 50, 'height':
6}
```

[32]:
```
#We can also add more than one key using the update() method.

person = {"name": "Jessa", 'country': "USA"}

# Adding 2 new keys at once pass new keys as dict
person.update({"weight": 50, "height": 6})
print(person)

#The item can be another dictionary or any iterable like a tuple of key-value␣
 ↪pairs.
# pass new keys as as list of tuple
person.update([("city", "Texas"), ("company", "Google",)])
print(person)
```

```
{'name': 'Jessa', 'country': 'USA', 'weight': 50, 'height': 6}
{'name': 'Jessa', 'country': 'USA', 'weight': 50, 'height': 6, 'city': 'Texas',
'company': 'Google'}
```

[33]:
```
#Set default value to a key
person_details = {"name": "Jessa", "country": "USA", "telephone": 1178}

# set default value if key doesn't exists
person_details.setdefault('state', 'Texas')

# key doesn't exists and value not mentioned. default None
person_details.setdefault("zip")

# key exists and value mentioned. doesn't  change value
person_details.setdefault('country', 'Canada')

# Display dictionary
for key, value in person_details.items():
    print(key, ':', value)
```

```
name : Jessa
country : USA
telephone : 1178
state : Texas
zip : None
```

Modify The Values Of The Dictionary Keys

We can modify the values of the existing dictionary keys using the following two ways :

Using key name We can directly assign new values by using its key name

Using update ( ) method: We can use the update method by passing the key-value pair to change the value.

[34]:
```
person = {"name": "Jessa", "country": "USA"}

# updating the country name
person["country"] = "Canada"
print(person['country'])

# updating the country name using update() method
person.update({"country": "USA"})
print(person['country'])
```

```
Canada
USA
```

Removing items from the dictionary

pop (key) - Return and removes the item with the key and return its value

popitem ( ) - Return and removes the last inserted item from the dictionary

del key - The del keyword will delete the item with the key that is passed

clear ( ) - Removes all items from the dictionary. Empty the dictionary

del dict_name - Delete the entire dictionary

[71]:
```
person = {'name': 'Jessa', 'country': 'USA', 'telephone': 1178, 'weight': 50,␣
 ↪'height': 6}

# Remove last inserted item from the dictionary
deleted_item = person.popitem()
print(deleted_item)  # output ('height', 6)
print(person)
print()


# Remove key 'telephone' from the dictionary
deleted_item = person.pop('telephone')
#deleted_item = person.pop('zip') --> Throws Error As Key Doesnt Exists
```

```
print(deleted_item)  # output 1178
print(person)
print()

# delete key 'weight'
del person['weight']
print(person)

# remove all item (key-values) from dict
person.clear()
print(person)  # {}
#person.popitem() - Throws Error As Dict is Empty

# Delete the entire dictionary
del person
print(person)
```

```
('height', 6)
{'name': 'Jessa', 'country': 'USA', 'telephone': 1178, 'weight': 50}

1178
{'name': 'Jessa', 'country': 'USA', 'weight': 50}

{'name': 'Jessa', 'country': 'USA'}
{}
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-71-547576c3a80f> in <module>
     26 # Delete the entire dictionary
     27 del person
---> 28 print(person)

NameError: name 'person' is not defined
```

Join two dictionary

We can add two dictionaries using :

The update ( ) method

Unpacking arbitrary keywords operator **

[43]:
```
#The dictionary to be added will be passed as the argument to the update()␣
 ↪method
#and the updated dictionary will have items of both the dictionaries.

dict1 = {'Jessa': 70, 'Arul': 80, 'Emma': 55}
```

```
dict2 = {'Kelly': 68, 'Harry': 50, 'Olivia': 66}

# copy second dictionary into first dictionary
dict1.update(dict2)
# printing the updated dictionary
print(dict1)
```

```
{'Jessa': 70, 'Arul': 80, 'Emma': 55, 'Kelly': 68, 'Harry': 50, 'Olivia': 66}
```

[44]:
```
#We can unpack any number of dictionary and add their contents to another␣
 ↪dictionary using **kwargs
student_dict1 = {'Aadya': 1, 'Arul': 2, }
student_dict2 = {'Harry': 5, 'Olivia': 6}
student_dict3 = {'Nancy': 7, 'Perry': 9}

# join three dictionaries
student_dict = {**student_dict1, **student_dict2, **student_dict3}
# printing the final Merged dictionary
print(student_dict)
```

```
{'Aadya': 1, 'Arul': 2, 'Harry': 5, 'Olivia': 6, 'Nancy': 7, 'Perry': 9}
```

[45]:
```
# Join two dictionaries having few keys in common
dict1 = {'Jessa': 70, 'Arul': 80, 'Emma': 55}
dict2 = {'Kelly': 68, 'Harry': 50, 'Emma': 66}

# join two dictionaries with some common items
dict1.update(dict2)
# printing the updated dictionary
print(dict1['Emma'])
```

```
66
```

Copy a Dictionary

Using copy ( ) method.

Using the dict ( )constructor

[46]:
```
dict1 = {'Jessa': 70, 'Emma': 55}

# Copy dictionary using copy() method
dict2 = dict1.copy()
print(dict2)

# Copy dictionary using dict() constructor
dict3 = dict(dict1)
print(dict3)

# Copy dictionary using the output of items() methods
```

```
dict4 = dict(dict1.items())
print(dict4)
```

```
{'Jessa': 70, 'Emma': 55}
{'Jessa': 70, 'Emma': 55}
{'Jessa': 70, 'Emma': 55}
```

[47]:
```
#Copy using the assignment operator
# When you set dict2 = dict1, you are making them refer to the same dict object

dict1 = {'Jessa': 70, 'Emma': 55}

# Copy dictionary using assignment = operator
dict2 = dict1

# modify dict2
dict2.update({'Jessa': 90})

print(dict2)
print(dict1)
```

```
{'Jessa': 90, 'Emma': 55}
{'Jessa': 90, 'Emma': 55}
```

Checking if a key exists

[48]:
```
person = {'name': 'Jessa', 'country': 'USA', 'telephone': 1178}

# Get the list of keys and check if 'country' key is present
key_name = 'country'
if key_name in person.keys():
    print("country name is", person[key_name])
else:
    print("Key not found")
```

```
country name is USA
```

Sort dictionary

[49]:
```
#The built-in method sorted() will sort the keys in the dictionary and returns␣
 ↪a sorted list

dict1 = {'c': 45, 'b': 95, 'a': 35}

# sort dict keys
print(sorted(dict1))

# sorting dictionary by keys
print(sorted(dict1.items()))
```

```
# sort dictionary values
print(sorted(dict1.values()))
```

```
['a', 'b', 'c']
[('a', 35), ('b', 95), ('c', 45)]
[35, 45, 95]
```

Python Built-in functions with dictionary

[50]:
```
#max() and min()
myDict = {1:'aaa',2:'bbb',3:'AAA'}
print('Maximum Key',max(myDict)) # 3
print('Minimum Key',min(myDict)) # 1
```

```
Maximum Key 3
Minimum Key 1
```

[52]:
```
#any() and all()

#dictionary with both 'true' keys
dict1 = {1:'True',1:'False'}

#dictionary with one false key
dict2 = {0:'True',1:'False'}

print('All True Keys : ',all(dict1))
print('One False Key with all : ',all(dict2))
print('One False Key with any : ',any(dict2))
```

```
All True Keys :  True
One False Key with all :  False
One False Key with any :  True
```

Dictionary Comprehension

[53]:
```
square_dict = dict()
for num in range(1, 11):
    square_dict[num] = num*num
print(square_dict)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

[54]:
```
# dictionary comprehension
square_dict = {num: num*num for num in range(1, 11)}
print(square_dict)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

[55]:
```
#item price in dollars
old_price = {'milk': 1.02, 'coffee': 2.5, 'bread': 2.5}

dollar_to_pound = 0.76
new_price = {item: value*dollar_to_pound for (item, value) in old_price.items()}
print(new_price)
```

```
{'milk': 0.7752, 'coffee': 1.9, 'bread': 1.9}
```

[56]:
```
#If Conditional Dictionary Comprehension
original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'john': 33}

even_dict = {k: v for (k, v) in original_dict.items() if v % 2 == 0}
print(even_dict)
```

```
{'jack': 38, 'michael': 48}
```

[57]:
```
#Multiple if Conditional Dictionary Comprehension
original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'john': 33}

new_dict = {k: v for (k, v) in original_dict.items() if v % 2 != 0 if v < 40}
print(new_dict)
```

```
{'john': 33}
```

[58]:
```
# if-else Conditional Dictionary Comprehension

original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'john': 33}

new_dict_1 = {k: ('old' if v > 40 else 'young') for (k, v) in original_dict.
 ↪items()}
print(new_dict_1)
```

```
{'jack': 'young', 'michael': 'old', 'guido': 'old', 'john': 'young'}
```

[59]:
```
# Nested Dictionary Comprehension
dictionary = {
    k1: {k2: k1 * k2 for k2 in range(1, 6)} for k1 in range(2, 5)
}
print(dictionary)
```

```
{2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}, 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15}, 4: {1:
4, 2: 8, 3: 12, 4: 16, 5: 20}}
```

[61]:
```
#the above code would be equivalent to:
dictionary = dict()
for k1 in range(2, 5):
    dictionary[k1] = {k2: k1*k2 for k2 in range(1, 6)}
print(dictionary)
```

```
{2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}, 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15}, 4: {1:
4, 2: 8, 3: 12, 4: 16, 5: 20}}
```

[62]:
```
#It can further be unfolded:
dictionary = dict()
for k1 in range(2,5):
    dictionary[k1] = dict()
    for k2 in range(1, 6):
        dictionary[k1][k2] = k1*k2
print(dictionary)
```

```
{2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}, 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15}, 4: {1:
4, 2: 8, 3: 12, 4: 16, 5: 20}}
```

When to use dictionaries ? :

Python dictionaries allow us to associate a value to a unique key, and then to quickly access this value. It's a good idea to use them whenever we want to find (lookup for) a certain Python object. We can also use lists for this scope, but they are much slower than dictionaries.

This speed is due to the fact that dictionary keys are hashable. Every immutable object in Python is hashable, so we can pass it to the hash() function, which will return the hash value of this object. These values are then used to lookup for a value associated with its unique key.

Python Interview Questions on Dictionaries

What is a Python dictionary ? How does it work ?

What's the difference between list.pop( ) and dictionary.pop( )

What python object do you cast to a dataframe ?

How dicts are better than lists ?

What are properties of python dicts ?

To-Do:

Write a Python function to interchange Keys and Values in a Dictionaries

Write a Python function to get min and max keys corresponding to min and max value in Dictionary

Write a Python function to remove none value items from Dictionary.

Write a Python function for Counting the frequencies of elements in a list or String using dictionary

Write a function in Python that takes a list of integers as a parameter and returns a dictionary whose keys are the list integers and whose values are "even" or "odd" depending on the number parity.

There are 36 possible combinations of two dice. A sim→ple pair of loops over range(6)+1 will enumerate all combinations. The sum of the two dice is more interesting than the actual combination. Create a dict of all combinations, using the sum of the two dice as the key.Each value in the dict should be a list of tuples; each tuple has the value of two dice

Write a Python function to Check if the Substring Matches Any Key in a Dictionary

Create a database in the following format :

	Interface	IP	status
1	Ethernet0	1.1.1.1	up
2	Ethernet1	2.2.2.2	down
3	Serial0	3.3.3.3	up
4	Serial1	4.4.4.4	up

Write a python program to find status of a given interface

Write a python program to find interface and IP of all interfaces which are up

Write a python program to count how many ethernet interfaces are there

Write a python program to add a new entry to above database

Write a Python Program to sort (ascending and descending) a dictionary by value.