12 May 2021 16:13

Janani Ravi

Working with dictionary & sets in Python

(1) Dictionary

Associative containers with key | value pairs

Keys unique, duplic vals ok

given key, finding val optimized by python

(2) Sets : Modeled on concept of sets in mathematics

no duplicates

12 May 2021 16:22

Introducing dictionaries

1. mapping of key-value pairs
2. Curly brackets
3. Record in dict is key-value mapping.
4. Records separated by commas
5. Key & values separated by comma
6. Keys may be either strings or ints
7. mydict.keys()
8. use "in" to check for presence of particular key
9. dict.values()
10. Can use mydict["key"] = "value" to add and update key-value pairs
11. Can also use .get method (

- can also use .get method to get value of key

11. Dictionaries are mutable
    - use del keyword to delete key:value pair
    - del mydict["key"] ////////

12 May 2021 16:21

Nesting Complex data types within dicts

1. dict = { } empty dict

2. dict.get( )

SYNTAX

dict.get(key , default=None)

Contrast with
dict['key'], which returns
KeyError if Key not present!

3. Values may be complex data types:
lists for example

4. mydict['data'][1]
mydict.get('data')[1]

13 May 2021 14:33

Invoking Functions on Dictionaries

(1) len(my_dict) // No. records

(2) len(mydict.keys())
len(mydict.values())

(3) sorted(my_dict)

A sorted dict

Returns a list of sorted keys

sorted(my_dict)
// ['dick', 'harry', 'tom']

(4) To sort my_dict

mydict_new = dict(sorted(mydict.items(), key = lambda x : x[0]))

```
mydict
{'tom': 'brown', 'dick': 'price', 'harry': 'worth'}

sorted(mydict)
['dick', 'harry', 'tom']

mydict.items()
dict_items([('tom', 'brown'), ('dick', 'price'), ('harry', 'worth')])

>>> sorted(mydict.items(), key=lambda x: x[0])
[('dick', 'price'), ('harry', 'worth'), ('tom', 'brown')]

>>> mydict_new=dict(sorted(mydict.items(), key=lambda x: x[0]))

>>> mydict
```

← note dict_items

← List of Tuples

```
>>> mydict_new=dict(sorted(mydict.items(), key=lambda x: x[0]))

>>> mydict
{'tom': 'brown', 'dick': 'price', 'harry': 'worth'}

>>> mydict_new
{'dick': 'price', 'harry': 'worth', 'tom': 'brown'}
```

sort a dictionary

(5) To sort values

sorted (my_dict.values())

(6) To copy a dictionary

mydict.copy() ← method

```
mydict_new_2=mydict.copy()

Mydict
 //   {'tom': 'brown', 'dick': 'price', 'harry': 'worth'}

mydict_new_2
 //  {'tom': 'brown', 'dick': 'price', 'harry': 'worth'}

mydict_new_2['mick']="miller"
mydict_new_2
 // {'tom': 'brown', 'dick': 'price', 'harry': 'worth', 'mick': 'miller'}

Mydict
 //  {'tom': 'brown', 'dick': 'price', 'harry': 'worth'}
```

(7) mydict.pop(1) Method

(a) Removes key and returns value

(b) modifies in place

(c) If key not there, get KeyError

```
 mydict.pop('tom')
'brown'
mydict
// {'dick': 'price', 'harry': 'worth'}
```

```
mydict
// {'dick': 'price', 'harry': 'worth'}
```

KeyError

(8) mydict.popitem()

arbitrarily removes a key-value pair (random)

(9) IF TWO identical keys in dict (literal last one prevails.

(10) dict.update() update keys

mydict.update(mydict_new).

keys updated with keys from new_dict

```
mydict
{'tom': 'brown', 'dick': 'price', 'harry': 'worth'}

mydict_new={'tom' : 'dunne'}

mydict.update(mydict_new)

mydict
{'tom': 'dunne', 'dick': 'price', 'harry': 'worth'}
```

(11) mydict.clear()

clears all key-value pairs to give empty dictionary

(12) to delete mydict completly, use del

del my-dict

16 May 2021 18:47

Introducing sets

(1) An unordered collection of unique immutable objects

(2) x = { } is empty dict
not empty set

(3) No duplicates

(4) No intrinsic order

(5) empty_set = set() // create empty set
single arg !!

(6) empty set displayed as set(), to distinguish from dictionaries which also use curly braces

```
x={}
type(x)
// <class 'dict'>

xx = set()
 type(xx)
<class 'set'>
x  //  {}
xx  //  set()
```

(7) Sets may contain

mixed data types

(8) Tuples may be elements of sets

(9) BUT lists may not be elements of sets!!

(10) set() takes a single optional argument

(11) A tuple may be dictionary key

(12) A set may not be a member of a set

(13) A dict key or set member must be hashable.

(14) Sets do not support indexing

(15) my_set.add(9) — Add to set

Set ← +, Add mnemonic!

(16)

mnemonic !

16 myset.update(my_tuple, my_list)

iterable objects (NOT int)

```
mylist =[1,2,3]
mytuple=[4,5,6,1]

myset.update(100, mylist, mytuple, 'hello')
Traceback (most recent call last):
  File "<pyshell#90>", line 1, in <module>
    myset.update(100, mylist, mytuple, 'hello')
TypeError: 'int' object is not iterable
```

← Note

```
myset.update(mylist, mytuple, 'hello')
myset   //   {1, 2, 3, 4, 5, 6, 'h', 'o', 'l', 'e'}
```

← NOTE

17 myset.clear() ✓

18 max(my_set) , min(my_set)

19 myset.discard('emma')

1) Takes 1 arg

2) NO error thrown if not there

20 also myset.remove()

but if element not there, throws key error

21 To Remove multiple values from set-

a my-set.difference_update({1,2,3}) //

Any iterable or Arg

(a) my_set.difference_update({1, 2, 3}) // any Arg

(b) my_set == {1, 2, 3} // must be set

NOTE my_set += {"ton"} will NOT work.

See:

https://stackoverflow.com/questions/49348340/how-to-remove-multiple-elements-from-a-set

18 May 2021 18:15

Performing set operations

1. set1.union(set2, set3) → Union creates new set

set1.union(set2,set3)

2. set1.intersection(set2, set3) → Find common elements

3. set1.difference(set2) // removes common elements

4. set1.intersection_update(set2)

takes intersection and sets equal to set1.

set1.intersection(set2) // {4, 5}

Set1 // {1, 2, 3, 4, 5}

set1.intersection_update(set2)

Set1 // {4, 5}

5. There is also difference_update

set1.difference_update(set2) //

6. set1.isdisjoint(set2)

Two sets are said to be disjoint if they have no elements in common (Alt, if intersection yields empty set)

7. set1.issubset(set2)

8. set1.issuperset(set2)

19 May 2021 09:17

Working with nested lists (matrices)

(1) [[a, b], [c, d], [e, f]]

↳ double square brackets

(2) len function gives length of outer list

```
mat=[[1,2,3],[4,5,6,7],[9]]
len(mat)   //   3
len(mat[1])   // 4
list(map(lambda x: len(x), mat))   //   [3, 4, 1]
[len(i) for i in mat]   //   [3, 4, 1]
```

(3) mat[0][1] // List-of-lists

(4) If either index out-of-range, Python throws IndexError

(5) slicing is possible, but slicing outer nested list gives nested list!!

```
mat=[[1,2,3],[4,5,6,7],[9]]
mat[1][0]   //   4
mat[1:][0]  //   [4, 5, 6, 7]
```

```
mat[0][1]     //   2
mat[0:][1]    //   [4, 5, 6, 7]

 mat[1:]      //   [[4, 5, 6, 7], [9]]
```

6

Why doesn't list have safe “get” method like dictionary?

From <https://stackoverflow.com/questions/5125619/why-doesnt-list-have-safe-get-method-like-dictionary>

But can use slicing with lists to avoid error message

19 May 2021 11:49

Performing list conversions

(1) tuple(my_list)

```
mlist =[['tom', 100], ['gerry', 95], ['emer', 99]]

 tuple(mlist)
(['tom', 100], ['gerry', 95], ['emer', 99])

[tuple(x) for x in mlist]
[('tom', 100), ('gerry', 95), ('emer', 99)]

(tuple(x) for x in mlist)
<generator object <genexpr> at 0x00000184A61B04A0>

tuple(tuple(x) for x in mlist)
(('tom', 100), ('gerry', 95), ('emer', 99))

map(lambda x: tuple(x), mlist)
<map object at 0x00000184A629BDC0>

tuple(map(lambda x: tuple(x), mlist))
(('tom', 100), ('gerry', 95), ('emer', 99))
```

(2) dict(my_list)

sub-elements must have length 2

```
mat =[['tom', 100], ['jerry', 200]]

dict(mat)
{'tom': 100, 'jerry': 200}

fname="tom", "gerry"
vals = 99, 9999

mydict2=dict(zip(fname, vals))

Mydict2
{'tom': 99, 'gerry': 9999}

>>> dict(fname)
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
```

```
>>> dict(fname)
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    dict(fname)
ValueError: dictionary update sequence element #0 has length 3; 2 is required
```

Note Value Error

Another example

```
mat_funny = [["tom",1001], ["mick", 1002, "b"]]
dict(mat_funny)
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    dict(mat_funny)
ValueError: dictionary update sequence element #1 has length 3; 2 is required

mat_funny_ok = [["tom",1001], ["mick", [1002, "b"]]]
dict(mat_funny_ok)   //   {'tom': 1001, 'mick': [1002, 'b']}
```

Nested list now OK

③ mylist.pop()

④ my_mat.clear

(but will not clear dict made from orig list)

⑤ list(my_dict) // gives a list of keys

⑥ list(my_dict.values()) // a list of vals

20 May 2021 08:33

Exercise Dictionary Sets

Dictionaries

(1) Unordered collection of key-value pairs

(2) Keys unique & hashable, values may be duplicates

(3) each key associated with value

(4) values looked up by key

(5) mapping only in 1 direction

(6) values may be complex data types

(7) Dictionaries are mutable

Sets

Unordered collection of unique elements.

(2) All elements must be hashable
- lists cannot be elements of sets, for eg
- neither can dicts

(3) Duplicates Auto removed

(4) union, intersection

Compare lists, dictionaries, sets

Compare lists, dictionaries, sets

(1) Lists ordered collection; both sets & dicts unordered.

(2) lists can contain duplicates, dicts cannot have duplicate keys
Duplicates ok for vals

(3) lists can have complex data types, so can vals in dictionary

(4) set elements must be immutable
eg immutable tuple & not other complex data types eg list & dict

(5) lists may be looked up by index. No index lookup for sets