11.4. Dictionary methods

Dictionaries have a number of useful built-in methods. The following table provides a summary and more details can be found in the Python Documentation.

Method	Parameters	Description
keys	none	Returns a view of the keys in the dictionary
values	none	Returns a view of the values in the dictionary
items	none	Returns a view of the key-value pairs in the dictionary
get	key	Returns the value associated with key; None otherwise
get	key,alt	Returns the value associated with key; alt otherwise

As we saw earlier with strings and lists, dictionary methods use dot notation, which specifies the name of the method to the right of the dot and the name of the object on which to apply the method immediately to the left of the dot. The empty parentheses in the case of `keys` indicate that this method takes no parameters. If `x` is a variable whose value is a dictionary, `x.keys` is the method object, and `x.keys()` invokes the method, returning a view of the value.

The keys method returns the keys, not necessarily in the same order they were added to the dictionary or any other particular order.

Save & Run Original - 1 of 1 Show in CodeLens

```
inventory = {'apples': 430, 'bananas': 312, 'oranges': 525, 'pears': 217}

for akey in inventory.keys():     # the order in which we get the keys is not defir
    print("Got key", akey, "which maps to value", inventory[akey])

ks = list(inventory.keys())
print(ks)
```

```
Got key apples which maps to value 430
Got key bananas which maps to value 312
Got key oranges which maps to value 525
Got key pears which maps to value 217
['apples', 'bananas', 'oranges', 'pears']
```

Activity: 1 -- ActiveCode (ac10_3_1)

It's so common to iterate over the keys in a dictionary that you can omit the `keys` method call in the `for` loop — iterating over a dictionary implicitly iterates over its keys.

Save & Run Original - 1 of 1 Show in CodeLens

```
inventory = {'apples': 430, 'bananas': 312, 'oranges': 525, 'pears': 217}

for k in inventory:
    print("Got key", k)

```

```
Got key apples
Got key bananas
Got key oranges
Got key pears
```

Activity: 2 -- ActiveCode (ac10_3_2)

The `values` and `items` methods are similar to `keys`. They return the objects which can be iterated over. Note that the item objects are tuples containing the key and the associated value.

Save & Run Original - 1 of 1 Show in CodeLens

```
inventory = {'apples': 430, 'bananas': 312, 'oranges': 525, 'pears': 217}

```

```
print(list(inventory.values()))
print(list(inventory.items()))

for k in inventory:
    print("Got",k,"that maps to",inventory[k])

```

```
[430, 312, 525, 217]
[('apples', 430), ('bananas', 312), ('oranges', 525), ('pears', 217)]
Got apples that maps to 430
Got bananas that maps to 312
Got oranges that maps to 525
Got pears that maps to 217
```

Activity: 3 -- ActiveCode (ac10_3_3)

Note

Technically, .keys(), .values(), and .items() don't return actual lists. Like the range function described previously, in python 3 they return objects that produce the items one at a time, rather than producing and storing all of them in advance as a list. Unless the dictionary has a whole lot of keys, this won't make a difference for performance. In any case, as with the range function, it is safe for you to think of them as returning lists, for most purposes. For the python interpreter built into this textbook, they actually do produce lists. In a native python interpreter, if you print out `type(inventory.keys())`, you will find that it is something other than an actual list. If you want to get the first key, `inventory.keys()[0]` works in the online textbook, but in a real python interpreter, you need to make the collection of keys into a real list before using `[0]` to index into it: `list(inventory.keys())[0]`.

The `in` and `not in` operators can test if a key is in the dictionary:

Save & Run Original - 1 of 1 Show in CodeLens

```
inventory = {'apples': 430, 'bananas': 312, 'oranges': 525, 'pears': 217}
print('apples' in inventory)
print('cherries' in inventory)

if 'bananas' in inventory:
    print(inventory['bananas'])
else:
    print("We have no bananas")

```

```
True
False
312
```

Activity: 4 -- ActiveCode (ac10_3_4)

This operator can be very useful since looking up a non-existent key in a dictionary causes a runtime error.

The `get` method allows us to access the value associated with a key, similar to the `[ ]` operator. The important difference is that `get` will not cause a runtime error if the key is not present. It will instead return None. There exists a variation of `get` that allows a second parameter that serves as an alternative return value in the case where the key is not present. This can be seen in the final example below. In this case, since "cherries" is not a key, return 0 (instead of None).

Save & Run Original - 1 of 1 Show in CodeLens

```
inventory = {'apples': 430, 'bananas': 312, 'oranges': 525, 'pears': 217}

print(inventory.get("apples"))
print(inventory.get("cherries"))

print(inventory.get("cherries",0))

```

```
430
None
0
```

Activity: 5 -- ActiveCode (ac10_3_5)

Check your understanding

dictionaries-3-1: What is printed by the following statements?

```
mydict = {"cat":12, "dog":6, "elephant":23, "bear":20}
answer = mydict.get("cat")//mydict.get("dog")
print(answer)
```

- (•) A. 2
- ( ) B. 0.5
- ( ) C. bear
- ( ) D. Error, divide is not a valid operation on dictionaries.

Check me | Compare me

✔ get returns the value associated with a given key so this divides 12 by 6.

Activity: 6 -- Multiple Choice (question10_3_1)

dictionaries-3-2: What is printed by the following statements?

```
mydict = {"cat":12, "dog":6, "elephant":23, "bear":20}
print("dog" in mydict)
```

- (•) A. True
- ( ) B. False

Check me | Compare me

✔ Yes, dog is a key in the dictionary.

Activity: 7 -- Multiple Choice (question10_3_2)

dictionaries-3-3: What is printed by the following statements?

```
mydict = {"cat":12, "dog":6, "elephant":23, "bear":20}
print(23 in mydict)
```

- ( ) A. True
- (•) B. False

Check me | Compare me

✔ Yes, the in operator returns True if a key is in the dictionary, False otherwise.

Activity: 8 -- Multiple Choice (question10_3_3)

dictionaries-3-4: What is printed by the following statements?

```
total = 0
mydict = {"cat":12, "dog":6, "elephant":23, "bear":20}
for akey in mydict:
   if len(akey) > 3:
      total = total + mydict[akey]
print(total)
```

- ( ) A. 18
- (•) B. 43
- ( ) C. 0
- ( ) D. 61

Check me | Compare me

✔ Yes, the for statement iterates over the keys. It adds the values of the keys that have length greater than 3.

Activity: 9 -- Multiple Choice (question10_3_4)

5. Every four years, the summer Olympics are held in a different country. Add a key-value pair to the dictionary `places` that reflects that the 2016 Olympics were held in Brazil. Do not rewrite the entire dictionary to do this!

Save & Run | 5/14/2021, 8:22:52 AM - 2 of 2 | Show in CodeLens

```
1 places = {"Australia":2000, "Greece":2004, "China":2008, "England":2012}
2
3 places["Brazil"]=2016
```

Activity: 10 -- ActiveCode (ac10_3_6)

Result	Actual Value	Expected Value	Notes
Pass	[('Au...004)]	[('Au...004)]	Testing that places has been updated correctly.

Expand Differences

You passed: 100.0% of the tests

6. We have a dictionary of the specific events that Italy has won medals in and the number of medals they have won for each event. Assign to the variable `events` a list of the keys from the dictionary `medal_events`. Do not hard code this.

Save & Run 5/14/2021, 8:23:13 AM - 2 of 2 Show in CodeLens

```
1 medal_events = {'Shooting': 7, 'Fencing': 4, 'Judo': 2, 'Swimming': 3, 'Diving': 2}
2
3 events=medal_events.keys()
```

Activity: 11 -- ActiveCode (ac10_3_7)

Result	Actual Value	Expected Value	Notes
Pass	['Div...ing']	['Div...ing']	Testing that events was created correctly

Expand Differences

You passed: 100.0% of the tests

11.3. Dictionary operations">
ctionary operations">
11.5. Aliasing and copying">
You have attempted 12 of 11 activities on this page

| Back to top