datagy

Python Merge Dictionaries – Combine Dictionaries (7 Ways)

November 18, 2021

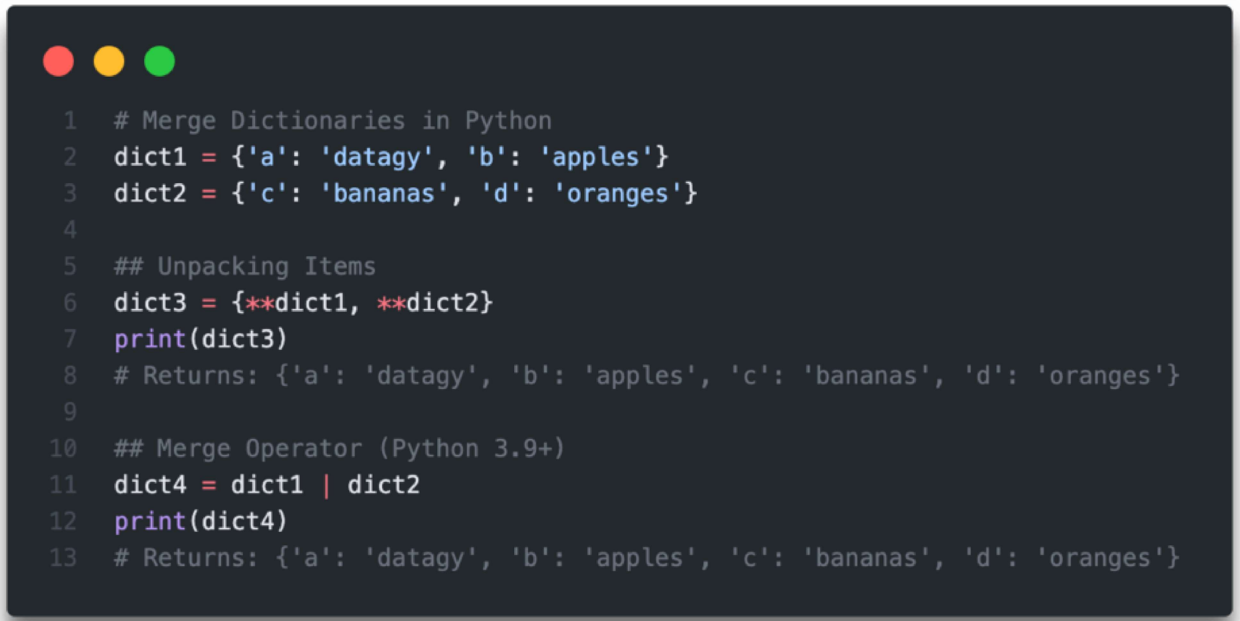
In this tutorial, **you'll learn how to use Python to merge dictionaries**. You'll learn how to combine dictionaries using different operators, as well as how to work with dictionaries that **contain the same keys**. You'll also learn how to append list values when merging dictionaries.

Python dictionaries are incredibly important data structures to learn. They are often used for their speedy data retrieval. Python dictionaries share many attributes with **JSON** format, which is often used in storing web data. When querying web APIs, you'll likely encounter data in JSON format. **Because of the important**

of retrieving web data, being able to combine dictionaries in Python is an important skill to understand.

Python dictionaries use a `key:value` mapping to store data. **Keys must be unique and must be immutable objects (such as strings or tuples).** Because of this, it's important to properly understand what will happen when you merge keys that share the same keys.

The Quick Answer: Use Unpacking or |



```python
# Merge Dictionaries in Python
dict1 = {'a': 'datagy', 'b': 'apples'}
dict2 = {'c': 'bananas', 'd': 'oranges'}

## Unpacking Items
dict3 = {**dict1, **dict2}
print(dict3)
# Returns: {'a': 'datagy', 'b': 'apples', 'c': 'bananas', 'd': 'oranges'}

## Merge Operator (Python 3.9+)
dict4 = dict1 | dict2
print(dict4)
# Returns: {'a': 'datagy', 'b': 'apples', 'c': 'bananas', 'd': 'oranges'}
```

Table of Contents

Merge Dictionaries in Python 3.9+

Python introduced a new way to merge dictionaries in Python 3.9, by using the merge operator `|`. Merging two dictionaries with the merge operator is likely the fastest and cleanest way to merge two dictionaries.

Let's see what this looks like in Python:

```python
# Merge two Python dictionaries using the merge operator in Python 3.9+
dict1 = {'a': 1, 'b': 2}
```

```
dict2 = {'c': 3, 'd': 4}

dict3 = dict1 | dict2

print(dict3)

# Returns: {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

We can see here that the two dictionaries have been merged successfully.

Like many other operators in Python, you can even use the `|=` operator combination to get the second dictionary to merge into the first without needing to reassign it.

Let's see what this looks like:

```
# Merge two Python dictionaries using the merge operator in Python 3.9+
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}

dict1 |= dict2

print(dict1)

# Returns: {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

We can see that this returned the same result as above.

Now let's take a look at an example of what happens when two dictionaries have a shared key.

```
# Merge two Python dictionaries using the merge operator in Python 3.9+
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'a': 4}

dict3 = dict1 | dict2

print(dict3)
```

```
# Returns: {'a': 4, 'b': 2, 'c': 3}
```

We can see that Python will **overwrite the left dictionary with the value of the keys of the right dictionary, if an overlap exists.** Keep this in mind! If you prefer to keep the values of the first dictionary, simply reverse the order in which you update the dictionaries.

In the next section, you'll learn how to use item unpacking to merge dictionaries in Python.

> **Want to learn how to pretty print a JSON file using Python?** Learn three different methods to accomplish this using this in-depth tutorial here.

Merge Python Dictionaries with Item Unpacking

You can also use item unpacking to merge Python dictionaries. The process of this involves adding every item from multiple dictionaries to a new dictionary.

Items in Python can be unpacked using either the `*` or the `**` characters. For dictionaries, to access both the key and value, you need to use the `**` characters.

Let's see how we can use this to merge two dictionaries in Python:

```
# Merge two Python dictionaries using item unpacking
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}

dict3 = {**dict1, **dict2}

print(dict3)

# Returns: {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

We can see that this successfully merges both dictionaries. **If you had more than two dictionaries, you could simply continue adding one another after another, separated by commas.**

What happens, though, if your dictionaries share keys? Let's take a look at what happens:

```
# Merge two Python dictionaries using item unpacking
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'a': 4}

dict3 = {**dict1, **dict2}

print(dict3)

# Returns: {'a': 4, 'b': 2, 'c': 3}
```

We can see that, similar to the merge operator method described above, the library on the right will overwrite the values of shared keys of the one on the left.

In the next section, you'll learn how to use the `.update()` method to merge dictionaries in Python.

> **Need to automate renaming files?** Check out this in-depth guide on using pathlib to rename files. More of a visual learner, the entire tutorial is also available as a video in the post!

Merge Python Dictionaries with the Update Method

The Python `.update()` method is used to, well, update items. In this case, we'll use the method to update dictionaries. Essentially, what it will do is update any existing key with new values or create new `key:value` pairs for non-existent keys.

Let's see how this works when all the keys are unique:

```
# Merge two Python dictionaries using .update()
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}

dict3 = dict1.copy()
dict3.update(dict2)

print(dict3)
```

```
# Returns: {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

We can see here that we first copied our first dictionary, then updated the dictionary. Keep in mind, this update happens in place, meaning that you don't need to reassign the dictionary to itself.

Let's now see what happens when the two dictionaries share a key:

```
# Merge two Python dictionaries using .update()
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'a': 4}

dict3 = dict1.copy()
dict3.update(dict2)

print(dict3)

# Returns: {'a': 4, 'b': 2, 'c': 3}
```

We can see that similar to the other two methods shown in the tutorial, the dictionary on the right will update any shared keys of the dictionary on the left.

In the next section, you'll see a more in-depth analysis of how merging two Python dictionaries with shared keys works.

> **Want to learn more about Python f-strings?** Check out my in-depth tutorial, which includes a step-by-step video to master Python f-strings!

Merge Python Dictionaries with Shared Keys

Python dictionaries require their keys to be unique. When you try to merge two or more dictionaries and there is overlap in keys, decisions need to be made as to which dictionary's values will be used for duplicate keys.

Python will always update records on the left with records on the right, regardless of what dictionary merging method you use (of the three identified above).

Intuitively, this makes the most sense when looking at the `.update()` method. Because the `.update()` method *updates* the values on the left dictionary with the values of the dictionary on the right, we can intuitively better understand this.

However, what happens when you merge more than two dictionaries, all with similar keys? **Python will continue to update the keys of the dictionary on the left, with the values of the dictionary on the right**.

Let's see what this looks like with the unpacking method:

```
# Merge three dictionaries with shared keys
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'a': 4}
dict3 = {'c': 3, 'a': 5}

dict4 = {**dict1, **dict2, **dict3}

print(dict4)

# Returns: {'a': 5, 'b': 2, 'c': 3}
```

We can see that while the key `a` is shared across all three dictionaries, on the value of the furthest right dictionary are kept for that key.

In the next section, you'll learn how to merge dictionaries by appending values for lists when duplicate keys exist.

> **Need to check if a key exists in a Python dictionary?** Check out this tutorial, which teaches you five different ways of seeing if a key exists in a Python dictionary, including how to return a default value.

Merge Python Dictionaries by Appending Values

There are many times that you may wish to store list values in your dictionary's values. When you then merge dictionaries, you may want to merge the lists, rather than overwriting them. That's exactly what you'll learn in this section!

For this, we'll loop over the keys and values of our dictionary that we want to merge into another. If the key exists, then we use the `.expand()` method to append to the list of the value.

Let's see what this looks like:

```
# Merge lists of values when merging dictionaries
dict1 = {'a': [1, 2, 3], 'b': [1, 2]}
dict2 = {'c': [4, 5, 6, 7], 'a': [4, 5, 6]}

for key, value in dict2.items():
    if key in dict1:
        dict1[key].extend(value)
    else:
        dict1[key] = value

print(dict1)

# Returns: {'a': [1, 2, 3, 4, 5, 6], 'b': [1, 2], 'c': [4, 5, 6, 7]}
```

In the next section, you'll learn some naive methods to merge Python dictionaries, beginning with a Python for loop.

> **Want to learn how to use the Python `zip()` function to iterate over two lists?** This tutorial teaches you exactly what the `zip()` function does and shows you some creative ways to use the function.

Merge Python Dictionaries with a For Loop

A naive implementation of merging dictionaries is to use a Python for loop.

For this method, we will loop over each `key:value` pair in a dictionary and see if it exists in the other. If the item exists, then we will update its value. If it doesn't, then we insert the new `key:value` pair.

Let's see what this looks like:

```
# Merge three dictionaries with shared keys
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'a': 4}

for key, value in dict2.items():
    dict1[key] = value

print(dict1)

# Returns: {'a': 4, 'b': 2, 'c': 3}
```

In the next section, you'll learn how to use a Python dictionary comprehension.

Want to learn more about Python for-loops? Check out my in-depth tutorial that takes your from beginner to advanced for-loops user! Want to watch a video instead? Check out my YouTube tutorial here.

Merge Python Dictionaries with a Dictionary Comprehension

Python dictionary comprehensions works very similar to for loops. If you want to learn more about Python dictionary comprehensions, check out my in-depth tutorial here.

Let's see how we can use a Python dictionary comprehension to merge two dictionaries:

```
# Merge three dictionaries with shared keys
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'a': 4}

dict3 = {k: v for d in [dict1, dict2] for k, v in d.items()}

print(dict3)

# Returns: {'a': 4, 'b': 2, 'c': 3}
```

> **Want to learn more about Python list comprehensions?** Check out this in-depth tutorial that covers off everything you need to know, with hands-on examples. More of a visual learner, check out my YouTube tutorial here.

Conclusion

In this tutorial, you learned how to use Python to merge two or more dictionaries. You learned how to do this using built-in methods, such as the merge operator `|` available in Python 3.9 and higher, the `.update()`