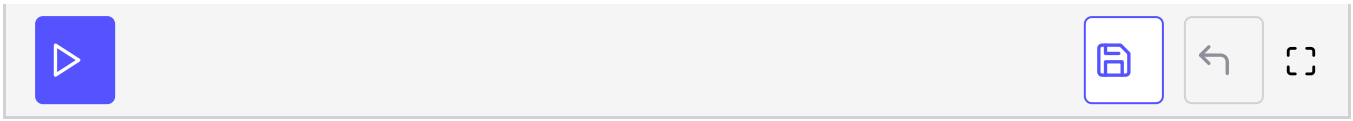
defaultdict

The collections module has a handy tool called **defaultdict**. The defaultdict is a subclass of Python's **dict** that accepts a default_factory as its primary argument. The default_factory is usually a Python type, such as int or list, but you can also use a function or a lambda too. Let's start by creating a regular Python dictionary that counts the number of times each word is used in a sentence:

```
sentence = "The red for jumped over the fence and ran to the zoo for food"
words = sentence.split(' ')

reg_dict = {}
for word in words:
    if word in reg_dict:
        reg_dict[word] += 1
    else:
        reg_dict[word] = 1

print(reg_dict)
```



If you run this code, you should see output that is similar to the following:

```
{'The': 1,
 'and': 1,
 'fence': 1,
 'food': 1,
 'for': 2,
 'jumped': 1,
 'over': 1,
 'ran': 1,
 'red': 1,
 'the': 2,
 'to': 1,
 'zoo': 1}
```

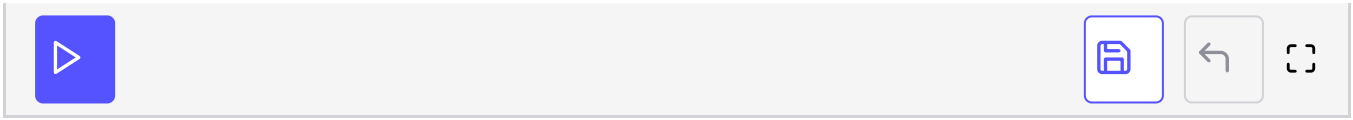
Now let's try doing the same thing with defaultdict!

```
from collections import defaultdict


sentence = "The red for jumped over the fence and ran to the zoo for food"
words = sentence.split(' ')

d = defaultdict(int)
for word in words:
    d[word] += 1

print(d)
```



You will notice right away that the code is much simpler. The defaultdict will automatically assign zero as the value to any key it doesn't already have in it. We add one so it makes more sense and it will also increment if the word appears multiple times in the sentence.

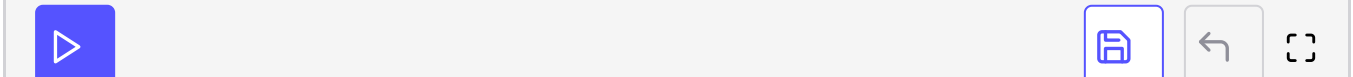
```
defaultdict(<class 'int'>,
            {'The': 1,
             'and': 1,
             'fence': 1,
             'food': 1,
             'for': 2,
             'jumped': 1,
             'over': 1,
             'ran': 1,
             'red': 1,
             'the': 2,
             'to': 1,
             'zoo': 1})
```

Now let's try using a Python list type as our default factory. We'll start off with a regular dictionary first, as before.

```
my_list = [(1234, 100.23), (345, 10.45), (1234, 75.00),
           (345, 222.66), (678, 300.25), (1234, 35.67)]

reg_dict = {}
for acct_num, value in my_list:
    if acct_num in reg_dict:
        reg_dict[acct_num].append(value)
    else:
        reg_dict[acct_num] = [value]

print(reg_dict)
```



This example is based on some code I wrote a few years ago. Basically I was reading a file line by line and needed to grab the account number and the payment amount and keep track of them. Then at the end, I would sum up each account. We're skipping the summing part here. If You run this code, you should get some output similar to the following:

```
{345: [10.45, 222.66], 678: [300.25], 1234: [100.23, 75.0, 35.67]}
```

Now let's re-implement this code using defaultdict:

```
from collections import defaultdict


my_list = [(1234, 100.23), (345, 10.45), (1234, 75.00),
           (345, 222.66), (678, 300.25), (1234, 35.67)]

d = defaultdict(list)
for acct_num, value in my_list:
    d[acct_num].append(value)

print(d)
```

Once again, this cuts out the if/else conditional logic and makes the code easier to follow. Here's the output from the code above:

```
defaultdict(<class 'list'>,
            {345: [10.45, 222.66],
             678: [300.25],
             1234: [100.23, 75.0, 35.67]})
```

This is some pretty cool stuff! Let's go ahead and try using a lambda too as our default_factory!

```
from collections import defaultdict
animal = defaultdict(lambda: "Monkey")
animal['Sam'] = 'Tiger'
print (animal['Nick'])
#Monkey

print (animal)
```

```
#defaultdict(<function <lambda> at 0x7f32f26da8c0>, {'Nick': 'Monkey', 'Sam': 'Tiger'})
```

Here we create a defaultdict that will assign ‘Monkey’ as the default value to any key. The first key we set to ‘Tiger’, then the next key we don’t set at all. If you print the second key, you will see that it got assigned ‘Monkey’. In case you haven’t noticed yet, it’s basically impossible to cause a KeyError to happen as long as you set the default_factory to something that makes sense. The documentation does mention that if you happen to set the default_factory to None, then you will receive a KeyError. Let’s see how that works:

```
from collections import defaultdict
x = defaultdict(None)
x['Mike']
#Traceback (most recent call last):
#  File "/usercode/__ed_file.py", line 3, in <module>
# x['Mike']
#KeyError: 'Mike'
```

In this case, we just created a very broken defaultdict. It can no longer assign a default to our key, so it throws a KeyError instead. Of course, since it is a subclass of dict, we can just set the key to some value and it will work. But that kind of defeats the purpose of the defaultdict.