[When to use return (yield something)?](https://stackoverflow.com/questions/42184576/when-to-use-return-yield-something)

For a long time I didn't know you can't put return in front of a yield statement. But actually you can:

def gen():

return (yield 42)

which is similar to

def gen():

yield 42

return

And the only usage I can think of is to attach sent value to StopIteration: [pep-0380](https://www.python.org/dev/peps/pep-0380/#proposal)

return expr in a generator causes StopIteration(expr) to be raised upon exit from the generator.

def gen():

return (yield 42)

g = gen()

print(next(g)) # 42

try:

g.send('AAAA')

except StopIteration as e:

print(e.value) # 'AAAA'

But this can be done using an extra variable too, which is more explicit:

def gen():

a = yield 42

return a

g = gen()

print(next(g))

try:

g.send('AAAA')

except StopIteration as e:

print(e.value) # 'AAAA'

So it seems return (yield xxx) is merely a syntactic sugar. Am I missing something?

ANSWER:

Inside a generator the expressions (yield 42) will yield the value 42, but it also returns a value which is either None, if you use next(generator) or a given value if you use generator.send(value).

So as you say, you could use an intermediate value to get the same behavior, not because this is syntactical sugar, but because the yield expressions is literally returning the value you send it.

You could equally do something like

def my\_generator():

return (yield (yield 42) + 10)

If we call this, using the sequence of calls:

g = my\_generator()

print(next(g))

try:

print('first response:', g.send(1))

print('Second response:', g.send(22))

print('third response:', g.send(3))

except StopIteration as e:

print('stopped at', e.value)

First we get the output of 42, and the generator is essentially paused in a state you could describe like: return (yield <Input will go here> + 10), If we then call g.send(1) we get the output 11. and the generator is now in the state: return <Input will go here>, then sending g.send(22) will throw a StopIteration(22), because of the way return is handled in generators. So you never get to the third send because of the exception.

I hope this example makes it a bit more apparent how yield works in generators and why the syntax return (yield something) is nothing special or exotic and works exactly how you'd expect it.

As for the literal question, when would you do this? Well when ever you want to yield something, and then later return a StopIteration echoing the input of the user sent to the generator. Because this is literally what the code is stating. I expect that such behavior is very rarely wanted.

Note: by returning a yield, you can chain generators.