Send

This lesson discusses sending values to a generator function.

Send

In the previous section, we formally introduced generators. Interestingly, we can also pass data to a generator function using the **`send()`** method defined on the associated generator object. The generator function can receive the value using the following syntax:

```
    item = yield
```

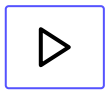
The **`send()`** method can also return a value to the caller. Consider the example program below:

```
def generator_function():
    while True:
        item = yield
        print("received " + str(item))


if __name__ == "__main__":
    gen = generator_function()
    next(gen)
    gen.send(37)
```


```
def generator_function():
    while True:
        item = yield
        print("received " + str(item))


if __name__ == "__main__":
    gen = generator_function()
    next(gen)
    gen.send(37)
```




Running the code above will produce an output of 37 in the console.

Note that before executing **`send()`** on the generator object we invoke **`next()`** one **line#9**. The reason is that when the generator is starting out, there is no **`yield`** assignment statement that the generator is waiting on to receive a value. If you comment out **line#9** you'll see an exception being thrown. But if you replace the value 37 being passed-in with None, the exception would go away. Another way we could have started the generator is to send a None value initially instead of invoking **`next()`**. The code change would be:

Starting a generator using send

```
def generator_function():
    while True:
        item = yield
        print("received " + str(item))


if __name__ == "__main__":
    gen = generator_function()
    # start a generator by sending None
    # which is not received but advances
    # the generator function to the first
    # yield statement
    gen.send(None)
    # send a value
    gen.send(37)
```

```
def generator_function():
    while True:
        item = yield
        print("received " + str(item))


if __name__ == "__main__":
    gen = generator_function()
    gen.send(None)
    gen.send(37)
```

When we execute `gen.send(37)` on **line#10** the generator function resumes execution and 37 is assigned to the variable `item`.

The `send()` method assigns the value within the generator function and then returns the next value yielded by the generator. If no value is yielded and the generator function exits, a `StopIteration` exception

is raised. Consider the following changes to the previous example and observe it throw an exception now.

```
def generator_function():
    item = yield
    print("received " + str(item))


if __name__ == "__main__":
    gen = generator_function()
    next(gen)
    gen.send(37)
```

```
def generator_function():
    item = yield
    print("received " + str(item))


if __name__ == "__main__":
    gen = generator_function()
    next(gen)
    gen.send(37)
```

Generators become interesting when we can send and receive from them. Let's try to combine the two functionalities in the next lesson.

Back
Generator

Next
Sending and Receiving

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/send__asyncio__python-concurrency-for-senior-engineering-interviews)