

Limited Period
Offer
Use Code:

Copy
Coupon

simplilearn

Other Segments

Yield in Python: An Ultimate Tutorial on Yield Keyword in Python

By Simplilearn

Last updated on Apr 14, 2021

321





Python has tons of utilities that make the lives of developers exponentially easier. One such utility is the yield keyword in Python, which can be used to replace return statements that you use in normal functions in Python. This comprehensive article will explore everything about the yield keyword in Python and how it is used in generator functions. So with no further ado, let's get started.

What Is Yield In Python?

The Yield keyword in Python is similar to a return statement used for returning values or objects in Python. However, there is a slight difference. The yield statement returns a generator object to the one who calls the function which contains yield, instead of simply returning a value.

Inside a program, when you call a function that has a yield statement, as soon as a yield is encountered, the execution of the function stops and returns an object of the generator to the function caller. In simpler words, the yield keyword will convert an expression that is specified along with it to a generator object and return it to the caller. Hence, if you want to get the values stored inside the generator object, you need to iterate over it.

It will not destroy the local variables' states. Whenever a function is called, the execution will start from the last yield expression. Please note that a function that contains a yield keyword is known as a generator function.

When you use a function with a return value, every time you call the function, it starts with a new set of variables. In contrast, if you use a generator function instead of a normal function, the execution will start right from where it left last.

If you want to return multiple values from a function, you can use generator functions with yield keywords. The yield expressions return multiple values. They return one value, then wait, save the local state, and resume again.

The general syntax of the yield keyword in Python is -

```
>>> yield expression
```

Before you explore more regarding yield keywords, it's essential first to understand the basics of generator functions.

Python Training Course

Learn Data Operations in Python

EXPLORE COURSE

Generator Functions In Python

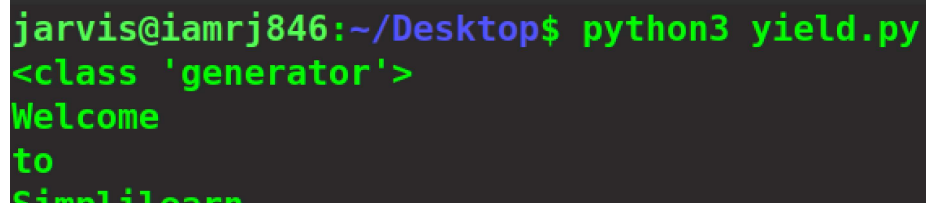
In Python, generator functions are those functions that, instead of returning a single value, return an iterable generator object. You can

In Python, generator functions are those functions that, instead of returning a single value, return an iterable generator object. You can access or read the values returned from the generator function stored inside a generator object one-by-one using a simple loop or using `next()` or `list()` methods.

You can create a generator function using the `generator()` and `yield` keywords. Consider the example below.

```
def generator():  
  
    yield "Welcome"  
  
    yield "to"  
  
    yield "Simplilearn"  
  
gen_object = generator()  
  
print(type(gen_object))  
  
for i in gen_object:  
  
    print(i)
```

In the above program, you have created a simple generator function and used multiple `yield` statements to return multiple values, which are stored inside a generator object when you create it. You can then loop over the object to print the values stored inside it.



```
jarvis@iamrj846:~/Desktop$ python3 yield.py  
<class 'generator'>  
Welcome  
to  
Simplilearn
```

A terminal window with a dark background. The prompt 'jarvis@iamrj846:~/Desktop\$' is shown in green and blue text, followed by a green cursor.

Let's create another generator function with yield keywords. You will try to filter out all the odd numbers from a list of numbers. Also, here it is essential to use different methods such as `list()`, `for-in`, and `next()` to output the values stored inside the generator object.

Consider the example below.

```
def filter_odd(numbers):
```

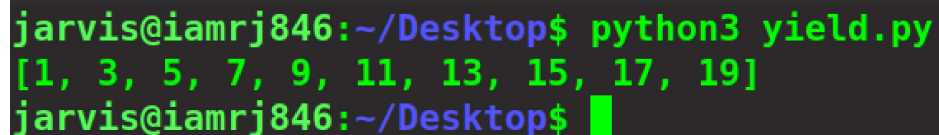
```
    for number in range(numbers):
```

```
        if(number%2!=0):
```

```
            yield number
```

```
odd_numbers = filter_odd(20)
```

```
print(list(odd_numbers))
```

A terminal window with a dark background. The prompt 'jarvis@iamrj846:~/Desktop\$' is shown in green and blue text. The command 'python3 yield.py' is entered in green text, followed by the output '[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]' in green text. The prompt is followed by a green cursor.

You can see that it has printed the generator object as a list.

...can see that it has printed the generator object as a new

You can also use the for-in loop to print the values stored inside the generator object. Here is how to do so.

```
def filter_odd(numbers):
```

```
    for number in range(numbers):
```

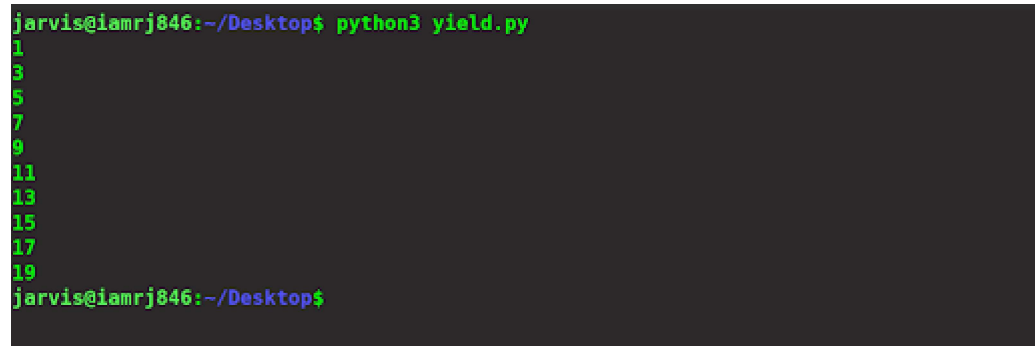
```
        if(number%2!=0):
```

```
            yield number
```

```
odd_numbers = filter_odd(20)
```

```
for num in odd_numbers:
```

```
    print(num)
```

A terminal window with a dark background and green text. The prompt is 'jarvis@iamrj846:~/Desktop\$'. The command 'python3 yield.py' has been executed, resulting in the output of the generator function: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19. The prompt is now 'jarvis@iamrj846:~/Desktop\$' again.

```
jarvis@iamrj846:~/Desktop$ python3 yield.py
1
3
5
7
9
11
13
15
17
19
jarvis@iamrj846:~/Desktop$
```

Finally, yet another method to print the elements stored inside a generator object is using the next() method. Each time you invoke the next() method on the generator object, it returns the next item.

```
def filter_odd(numbers):
```

```
def filter_odd(numbers):
```

```
    for number in range(numbers):
```

```
        if(number%2!=0):
```

```
            yield number
```

```
odd_numbers = filter_odd(20)
```

```
print(next(odd_numbers))
```

```
print(next(odd_numbers))
```

```
print(next(odd_numbers))
```

```
print(next(odd_numbers))
```

```
print(next(odd_numbers))
```

```
print(next(odd_numbers))
```

```
jarvis@iamrj846:~/Desktop$ python3 yield.py
```

```
1
```

```
3
```

```
5
```

```
7
```

```
9
```

```
11
```

```
jarvis@iamrj846:~/Desktop$
```

Please note that if there is no item left in the generator object and you invoke the `next()` method on it, it will return a `StopIteration` error.

Also, it's very important to note that you can call the generators only once in the same program. Consider the program below.

```
def filter_odd(numbers):
```

```
    for number in range(numbers):
```

```
        if(number%2!=0):
```

```
            yield number
```

```
odd_numbers = filter_odd(20)
```

```
print(list(odd_numbers))
```

```
for i in odd_numbers:
```

```
    print(i)
```

```
jarvis@iamrj846:~/Desktop$ python3 yield.py
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
jarvis@iamrj846:~/Desktop$
```


You can see that first when you invoked the list method on the generator object, it returned the output. However, next time, when you used the for-in loop to print the values, it returned nothing. Hence, you can conclude that you can use the generator objects only once. If you want to use it again, you need to call it again.

Example of Using Yield In Python (Fibonacci Series)

Here is a general example that you can use to understand the concept of yield in the most precise manner. Here is a Fibonacci program that has been created using the yield keyword instead of return.

```
def fibonacci(n):  
  
    temp1, temp2 = 0, 1  
  
    total = 0  
  
    while total < n:  
  
        yield temp1  
  
        temp3 = temp1 + temp2  
  
        temp1 = temp2  
  
        temp2 = temp3  
  
        total += 1
```

```
fib_object = fibonacci(20)
```

```
print(list(fib_object))
```

Here, you have created a Fibonacci program that returns the top 20 Fibonacci numbers. Instead of storing each number in an array or list and then returning the list, you have used the yield method to store it in an object which saves a ton of memory, especially when the range is large.

```
jarvis@iamrj846:~/Desktop$ python3 yield.py  
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]  
jarvis@iamrj846:~/Desktop$
```

How Can You Call Functions Using Yield?

Instead of return values using yield, you can also call functions. For example, suppose you have a function called cubes which takes an input number and cubes it, and there exists another function that uses a yield statement to generate cubes of a range of numbers. In such a case, you can use the cubes function along with the yield statement to create a simple program. Let's check out the code below.

```
def cubes(number):
```

```
    return number*number*number
```

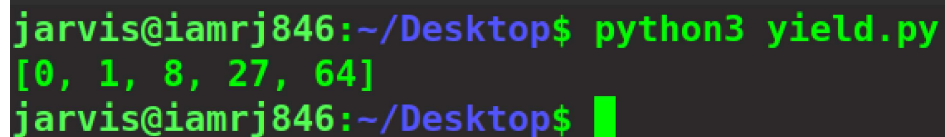
```
def getCubes(range_of_nums):
```

```
    for i in range(range_of_nums):
```

```
        yield cubes(i)
```

```
cube_object = getCubes(5)
```

```
print(list(cube_object))
```

A terminal window with a dark background and green text. The prompt is 'jarvis@iamrj846:~/Desktop\$'. The command 'python3 yield.py' is entered, followed by the output '[0, 1, 8, 27, 64]'. The prompt is then shown again with a cursor.

```
jarvis@iamrj846:~/Desktop$ python3 yield.py  
[0, 1, 8, 27, 64]  
jarvis@iamrj846:~/Desktop$
```

You can see how you can use yield to compute values by calling the function directly along with the statement and store them in a generator object.

Why And When Should You Use Yield?

When you use a yield keyword inside a generator function, it returns a generator object instead of values. In fact, it stores all the returned values inside this generator object in a local state. If you have used the return statement, which returned an array of values, this would have consumed a lot of memory. Hence, yield should always be preferred over the return in such cases.

Moreover, the execution of the generator function starts only when the caller iterates over the generator object. Hence, it increases the

overall efficiency of the program along with decreasing memory consumption. Some situations where you should use yield are -

1. When the size of returned data is quite large, instead of storing them into a list, you can use yield.
2. If you want faster execution or computation over large datasets, yield is a better option.
3. If you want to reduce memory consumption, you can use yield.
4. It can be used to produce an infinite stream of data. You can set the size of a list to infinite, as it might cause a memory limit error.
5. If you want to make continuous calls to a function that contains a yield statement, it starts from the last defined yield statement, and hence, you can save a lot of time.

Yield Vs. Return In Python

Before you understand the difference between yield and return in Python, it's very important to understand the differences between a normal function that uses a return statement and a generator function that uses a yield statement.

A normal function directly stores and returns the value. However, generator functions return generator objects which contain all the values to be returned and they store them locally, thus reducing a lot of memory usage.

Also, when you call a normal function, the execution stops as soon as it gets to the return statement. Hence, after starting, you can't stop the execution of a normal function. However, in the case of generator functions, as soon as it reaches the first yield statement, it stops the execution and sends the value to the generator function. When the caller iterates over this value, then the next yield statement is processed, and the cycle continues.

Below are some differences between yield and return in Python.

Yield	Return
When the caller calls the generator function, it packs all the return values from yield into a generator object and returned. Also, the code execution starts only when the caller iterates over the object.	It returns only a single value to the caller, and the code execution stops as soon as it reaches the return statement.
When a caller calls the generator function, the first yield is executed, and the function stops. It then returns the generator object to the caller where the value is stored. When the caller has accessed or iterated over this value, then the next yield statement is executed and the cycle repeats.	When the caller calls a normal function, the execution begins and ends as soon as it reaches a return statement. It then returns the value to the caller.
You can use multiple yield statements in a generator function.	Only one return statement in a normal function can be used.
There is no memory allocation when you use yield keywords.	For all the returned values, memory is allocated.
Extremely memory-efficient, especially dealing with large data sets.	Should be used only with small data sets.
For large data sets, execution time is faster when the yield keyword is used.	More execution time since extra processing

For large data sets, execution time is faster when the yield keyword is used.

has to be done if the data size is large.

FREE Data Science With Python Course

Start Learning Data Science with Python for FREE

START LEARNING

Advantages And Disadvantages of Yield

The advantages of using yield keywords instead of return are that the values returned by yield statement are stored as local variables states, which allows control over memory overhead allocation. Also, each time, the execution does not start from the beginning, since the previous state is retained.

However, a disadvantage of yield is that, if the calling of functions is not handled properly, the yield statements might sometimes cause errors in the program. Also, when you try to use the yield statements to improve time and space complexities, the overall complexity of the code increases which makes it difficult to understand.

Looking forward to making a move to the programming field? Take up the [Python Training Course](#) and begin your career as a professional Python programmer

Wrapping Up!

To sum up, you can leverage the yield statements in Python to return multiple values from generator functions. It is highly memory-efficient and increases the overall performance of the code. It saves memory because it stores the values to be returned as local variables state, and also each time it executes the function, it need not start from the beginning as the previous states are retained. This is what makes yield keywords highly popular among developers and a great alternative to return statements.

In this tutorial, you explored how you can leverage yield in Python to optimize programs in terms of both speed and memory. You saw several examples of generator functions and the different scenarios where you can use the yield statements. Moreover, you also explored why and when should you use it, along with its advantages and disadvantages.

You differentiated the use of normal functions and generator functions, and at the same time, you also compared return statements with yield keywords.

We hope that this comprehensive tutorial will give you better in-depth insights into yield keywords in Python.

If you are looking to learn further and master python and get started on your journey to becoming a Python expert, Simplilearn's [Python Certification Course](#) should be your next step. This comprehensive course gives you the work-ready training you need to master python including key topics like [data operations](#), shell scripting, and conditional statement. You even get a practical hands-on exposure to Djang in this course.

If on the other hand, you have any queries or feedback for us on this yield in python article, do mention them in the comments section at the end of this page. We will review them and respond to you at the earliest.

Happy Learning!

Find our Python Training Online Classroom training classes in top cities:

Name	Date	Place	
Python Training	5 Jun -27 Jun 2021, Weekend batch	Your City	View Details

About the Author

 [Simplilearn](#)

Simplilearn is one of the world's leading providers of online training for Digital Marketing, Cloud Computing, Project Management, Data Science, IT, Software Development, and many other emergi...

[View More](#)

Recommended Programs

Python Training

2723 Learners

Lifetime
Access*

Post Graduate Program in Data Science

1936 Learners

Lifetime
Access*

*Lifetime access to high-quality, self-paced e-learning content.

Explore Category

NEXT ARTICLE

What is Pass in Python and Usage of Pass Function in Python

By Simplilearn

74

Mar 29, 2021

Recommended Resources

Python Interview Guide

What is Python Used For? Top
5 Use Cases E...

0 Comments [Simplilearn](#) [Disqus' Privacy Policy](#)

 **Login** ▼

 **Recommend**  **Tweet**  **Share**

Sort by Best ▼



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name

Be the first to comment.

 **Subscribe**  **Add Disqus to your site** [Add DisqusAdd](#)  **Do Not Sell My Data**

Disclaimer

PMP, PMI, PMBOK, CAPM, PgMP, PfMP, ACP, PBA, RMP, SP, and OPM3 are registered marks of the Project Management Institute, Inc.