Generators in Python

Difficulty Level : Easy • Last Updated : 31 Mar, 2020

Prerequisites: Yield Keyword and Iterators

There are two terms involved when we discuss generators.

Data Structures Algorithms Interview Preparation Topic-wise Practice C++ Java Python

1. **Generator-Function :** A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. If the body of a def contains yield, the function automatically becomes a generator function.

```python
# A generator function that yields 1 for first time,
# 2 second time and 3 third time
def simpleGeneratorFun():
    yield 1
    yield 2
    yield 3

# Driver code to check above generator function
for value in simpleGeneratorFun():
```

2. **Generator-Object :** Generator functions return a generator object. Generator objects are used either by calling the next method on the generator object or using the generator object in a “for in” loop (as shown in the above program).

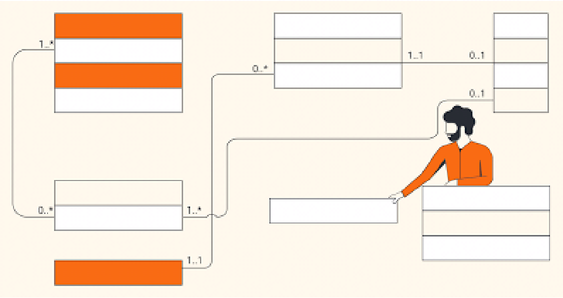
```
# A Python program to demonstrate use of
# generator object with next()

# A generator function
def simpleGeneratorFun():
    yield 1
    yield 2
    yield 3

# x is a generator object
x = simpleGeneratorFun()

# Iterating over the generator object using next
print(x.next()) # In Python 3, __next__()
print(x.next())
print(x.next())
```

Output :



So a generator function returns an generator object that is iterable, i.e., can be used as an Iterators .

As another example, below is a generator for Fibonacci Numbers.

```python
# A simple generator for Fibonacci Numbers
def fib(limit):

    # Initialize first two Fibonacci Numbers
    a, b = 0, 1

    # One by one yield next Fibonacci Number
    while a < limit:
        yield a
        a, b = b, a + b

# Create a generator object
x = fib(5)

# Iterating over the generator object using next
print(x.next()) # In Python 3, __next__()
print(x.next())
print(x.next())
print(x.next())
print(x.next())

# Iterating over the generator object using for
# in loop.
print("\nUsing for in loop")
for i in fib(5):
    print(i)
```

Output :

```
0
1
```

```
1
2
3
```

Applications : Suppose we to create a stream of Fibonacci numbers, adopting the generator approach makes it trivial; we just have to call next(x) to get the next Fibonacci number without bothering about where or when the stream of numbers ends.
A more practical type of stream processing is handling large data files such as log files. Generators provide a space efficient method for such data processing as only parts of the file are handled at one given point in time. We can also use Iterators for these purposes, but Generator provides a quick way (We don't need to write __next__ and __iter__ methods here).

Refer below link for more advanced applications of generators in Python.
http://www.dabeaz.com/finalgenerator/

This article is contributed by **Shwetanshu Rohatgi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Like 96

Previous

Next

RECOMMENDED ARTICLES

Page : 1 2 3 4 5 6

01 Using Generators for substantial memory savings in Python
24, Apr 20

02 CNN - Image data pre-processing with generators
15, Jul 20

03 Important differences between Python 2.x and Python 3.x with examples
25, Feb 16

04 Python | Set 4 (Dictionary, Keywords in Python)
09, Feb 16

Article Contributed By :

Improved By : fridayda13

Article Tags : Python

Improve Article

Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us

Careers

Privacy Policy

Learn

Algorithms

Data Structures

Languages

CSS

JavaScript

Bootstrap

Internships

Videos