PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Generators

Prof. Sindhu R Pai
PCPS Theory Anchor - 2024
Department of Computer Science and Engineering

Functions - Generators

- A generator is a function that **returns an iterator that produces a sequence of values** when iterated over.
- **Iterator** - an object that can **be iterated upon**, i.e. we can traverse through all the values.
- Generators in Python provides a way to create a function that behaves like an iterator.
- Does not return a single value; instead, it **returns an iterator object with a sequence of values**.
- **A *yield* statement is used** instead of the *return* statement.
- If the body of a def contains *yield*, the function **automatically becomes a Python generator function.**

Functions - Generators

- **Syntax**

```
def generator_function_name(arg):
                ………………………
                 ………………………
                  ………………………
                  yield statement
```

- When the generator function is called, it does not execute the function body immediately. Instead, it returns a generator object that can be iterated over to produce the values.

Generator Object

- Python Generator functions return a generator object that is iterable (used as an iterator).

- Generator objects are accessed
  - by calling the next method of the generator object (Refer Example 2)

    or
  - using the generator object in a “for” loop (Refer Example 1)

Functions - Generators

Example 1: Simple generator function that will yield three integers (using for loop)

```
# Generator function
def generator_func():
    yield 1
    yield 2
    yield 3

# Code to check above generator function
for value in generator_func():
    print(value)
```

Output:

```
1
2
3
```

Example 2: Simple generator function that will yield three integers (using next() function)

```
# Generator function
def generator_func():
    yield 10
    yield 20
    yield 30

#obj is a generator object
obj=generator_func()

# Iterating over the generator object using next
print(next(obj))
print(next(obj))
print(next(obj))
```

Output:

```
10
20
30
```

Generator Expression

- Generator expression is another way of writing the generator function.
- Similar to list comprehension technique but instead of storing the elements in a list in memory, it creates generator objects.
- **Syntax**:

```
(expression for element in iterable)
```

Generator Expression - Example

```
#Generator Expression
generator_exp=(i**2 for i in range(5) if i%2==0)

for i in generator_exp:
    print(i)
```

Output:

```
0
4
16
```

Pipelining Generators

Multiple generators can be used to pipeline a series of operations

Example: Compute the sum of squares of numbers in the Fibonacci series

```
# Generator function - fibonacci_numbers
def fibonacci_numbers(nums):
    x,y=0,1
    for i in range(nums):
        x,y=y,x+y
        yield x
# Generator function - square
def square(nums):
    for num in nums:
        yield num**2
print(sum(square(fibonacci_numbers(3))))
```

Output:

```
6
```

PES UNIVERSITY

Function Generators: yield vs. return

yield	return
Returns a value and pauses the execution while maintaining the internal states	Returns a value and terminates the execution of the function
Used to convert a regular Python function into a generator	Used to return the result to the caller statement
Used when the generator returns an intermediate result to the caller	Used when a function is ready to send a value
Code written after yield statement execute in next function call	Code written after return statement won’t execute
It can run multiple times	It only runs a single time

Note: We can’t include *return* inside a generator function. If we do, it will terminate the function.

Function Generators: Summary

- Python generator functions allows for the declaration of a function that behaves like an iterator, making it a faster, cleaner and easier way to create an iterator.

- Generators are useful when we want to produce a large sequence of values, but we don't want to store all of them in memory at once.

- The simplification of code is a result of generator function and generator expression support provided by Python.

THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CDSAML & CCBD, PESU
Prof. Sindhu R Pai – sindhurpai@pes.edu
Prof. Sowmya Shree P