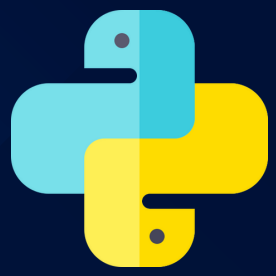
What are Generators in Python ?



SWIPE

JOGENDRA SINGH

What are generators?

Generators are objects that are lazy iterators. Like lists, we can iterate them, but they don't store value in memory. They generate the data next when required. Generators can be used to create an infinite sequence or iterate through a large piece of data. Generators objects can be made in 2 ways :

1. Functions
2. Expressions

SWIPE

Example

```python
# 2. Generator Expression
g_expression = (num for num in range(3,6))
# 1. Generator Function
def g_function(num):
    for i in range(0,num):
        yield i

Generator = g_function(3)
print(next(Generator)) #>> 0
for i in Generator:
    print("Generator : ",i) #>> 1 2
print("----")

print(next(g_expression)) #>> 3
for i in g_expression:
    print("Expression : ",i) #>> 4 5
```

SWIPE

Use of yield and next()

Yield's primary job is to control the flow of a generator function in a similar way to return statements.

When you call a generator function or use a generator expression, you return a special iterator called a generator.

When you call special methods on the generator, such as next(), the code within the function is executed up to **yield**.

SWIPE ▶

Usage

- Reading large data
- Creating Infinite iterable

SWIPE

Example

```python
# Generator for reading data from a file.
def csv_reader(file_name):
    for row in open(file_name, "r"):
        yield row

# Generator for infinite sequence.
def infinite_sequence():
    num = 0
    while True:
        yield num
        num += 1
```

SWIPE

Advantage

Saves memory

Yield only executes the statement below it and maintains local memory for variables.

Solves MemoryError

SWIPE ▶

Disadvantage

CPU heavy.

It consumes more time to loop through than the list counterpart.

SWIPE

Advance Methods

- .send( ) allows you to send value back to generator.
- .throw( ) allows you to throw exceptions with the generator.
- .close( ) allows you to stop a generator.

THANK YOU :)

JOGENDRA SINGH