# **🔁 Python Guide: Generator Functions using yield**

## **✅ What is a Generator Function?**

A **generator function** is a special function that **returns values one at a time** using the yield keyword  
 instead of returning everything at once with return.

🧠 It creates a **lazy sequence** — perfect for memory efficiency and large datasets.

## **🧪 Basic Syntax**

def get\_numbers(n):

for i in range(n):

yield i

* ✅ yield pauses the function and returns one value at a time
* ✅ On the next call, it resumes **right after the last yield**

## **🔍 How to Use It**

### **🔹 Option 1: for loop**

for num in get\_numbers(5):

print(num)

🔸 Output:

0

1

2

3

4

### **🔹 Option 2: Convert to list()**

print(list(get\_numbers(5)))

# Output: [0, 1, 2, 3, 4]

### **🔹 Option 3: Manual next()**

gen = get\_numbers(3)

print(next(gen)) # 0

print(next(gen)) # 1

print(next(gen)) # 2

After it finishes, calling next() again will raise:

StopIteration

## **🧠 Why Use Generators Instead of return?**

| **Feature** | **return** | **yield** |
| --- | --- | --- |
| Memory usage | Stores full result in memory | Streams one item at a time |
| Suitable for big data | ❌ Risky | ✅ Efficient |
| Execution style | Runs all at once | Runs step-by-step |
| Use in pipelines | ❌ Not flexible | ✅ Perfect for pipelines and loops |

## **📦 Real-Life Use Case: Reading Large Files**

def read\_lines(filename):

with open(filename) as f:

for line in f:

yield line.strip()

for line in read\_lines("bigfile.txt"):

print(line)

✅ Reads one line at a time — perfect for **log files, reports, big datasets**

## **💥 Bonus Example: Custom Range Generator**

def custom\_range(start, end, step):

while start < end:

yield start

start += step

for i in custom\_range(10, 20, 3):

print(i)

✅ Output:

10

13

16

19

## **⚠️ Common Mistake**

def wrong\_gen():

yield 1

return 2 # ❌ Wrong: `return` stops the generator

✅ Always use yield to continue producing values  
 ❌ Don’t mix yield and return unless you're done generating

## **✅ TL;DR (One-Liner):**

A **generator function** uses yield to return one value at a time  
 and **remembers its state** between calls — making it **efficient for big data and pipelines**.

## **🎯 Mini Practice for Your Viewers:**

def even\_numbers(n):

for i in range(n + 1):

if i % 2 == 0:

yield i

print(list(even\_numbers(10))) # [0, 2, 4, 6, 8, 10]

### **About the Author**

**Gowtham SB** is a **Data Engineering expert, educator,** **and content creator** with a passion for **big data technologies, as well as cloud and Gen AI** . With years of experience in the field, he has worked extensively with **cloud platforms, distributed systems, and data pipelines**, helping professionals and aspiring engineers master the art of data engineering.

Beyond his technical expertise, Gowtham is a **renowned mentor and speaker**, sharing his insights through engaging content on **YouTube and LinkedIn**. He has built one of the **largest Tamil Data Engineering communities**, guiding thousands of learners to excel in their careers.

Through his deep industry knowledge and hands-on approach, Gowtham continues to **bridge the gap between learning and real-world implementation**, empowering individuals to build **scalable, high-performance data solutions**.

𝐒𝐨𝐜𝐢𝐚𝐥𝐬

🎥𝐘𝐨𝐮𝐓𝐮𝐛𝐞 - https://www.youtube.com/@dataengineeringvideos

📸𝐈𝐧𝐬𝐭𝐚𝐠𝐫𝐚𝐦 - <https://instagram.com/dataengineeringtamil>

📸𝐈𝐧𝐬𝐭𝐚𝐠𝐫𝐚𝐦 - [https://instagram.com/](https://instagram.com/dataengineeringtamil)thedatatech.in

🤝𝐂𝐨𝐧𝐧𝐞𝐜𝐭 𝐟𝐨𝐫 𝟏:𝟏 - https://topmate.io/dataengineering/

💼𝐋𝐢𝐧𝐤𝐞𝐝𝐈𝐧 - https://www.linkedin.com/in/sbgowtham/

🌐𝐖𝐞𝐛𝐬𝐢𝐭𝐞 - https://codewithgowtham.blogspot.com

💻𝐆𝐢𝐭𝐇𝐮𝐛 - http://github.com/Gowthamdataengineer

💬𝐖𝐡𝐚𝐭𝐬 𝐀𝐩𝐩 - https://lnkd.in/g5JrHw8q

📧𝐄𝐦𝐚𝐢𝐥 - atozknowledge.com@gmail.com

📱𝐀𝐥𝐥 𝐌𝐲 𝐒𝐨𝐜𝐢𝐚𝐥𝐬 - <https://lnkd.in/gf8k3aCH>