

# Python Generator Genius

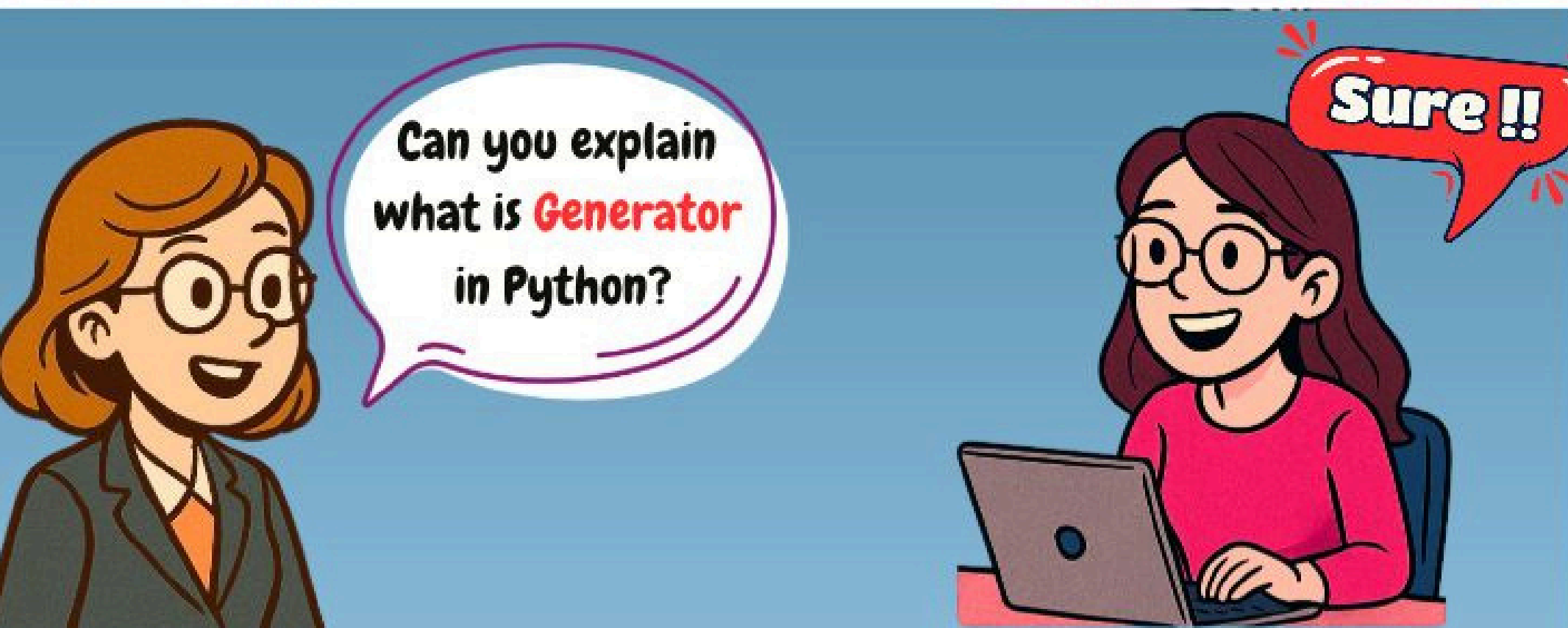


**\*Interviewer**



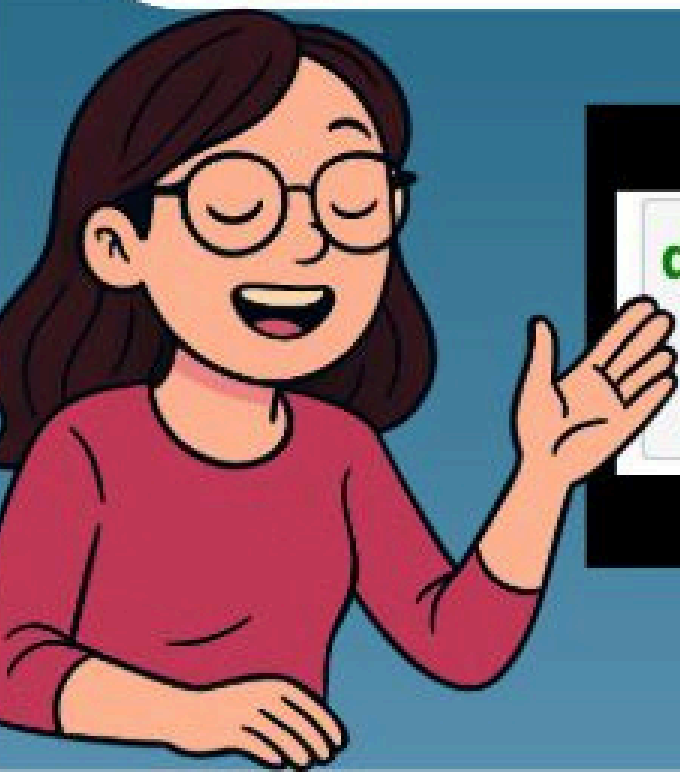
**\*Bhawana**

learnwithbhawana



learnwithbhawana

- A Generator is a special type of Iterable, like Lists or Tuples.
- Generators create Values **On-the-fly** using **yield**.



```
def generator_function():  
    # Logic  
    yield value
```

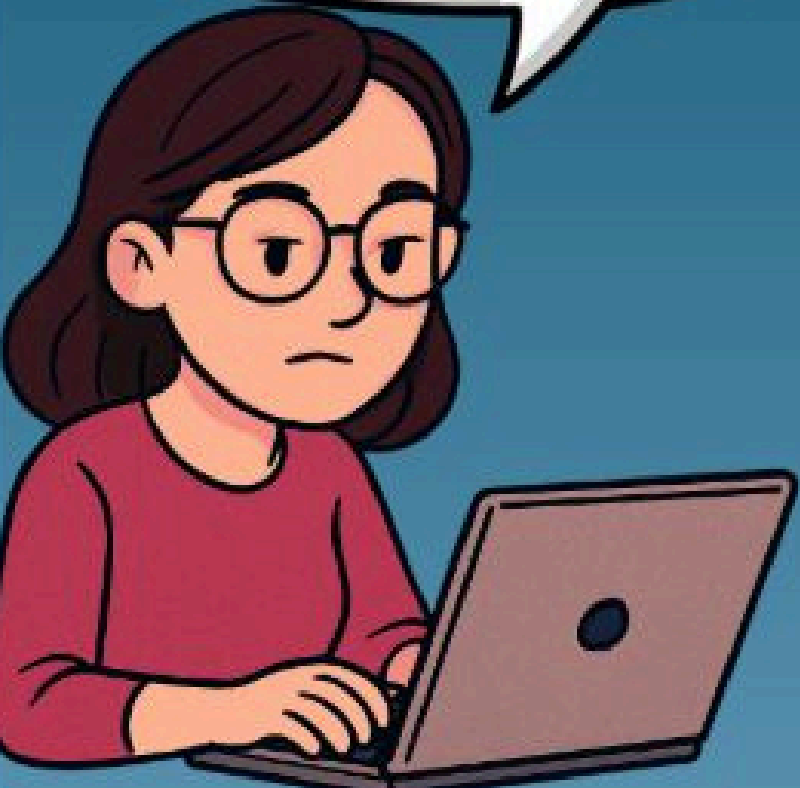


Correct.  
Can you give an example?



**EXAMPLE**





Here's a simple generator  
to generate numbers  
from 1 to 5

Code

```
def gen_numbers(n):  
    for i in range(1,n+1):  
        yield i  
  
#Using the generator  
  
for num in gen_numbers(5):  
    print(num)
```

O/p:

```
1  
2  
3  
4  
5
```



Great Job!!  
**Now**, What's the  
difference b/w  
**yield and return?**

yield  
return



## return

- terminates function execution
- returns a value(**cannot be resumed**)

## yield


- Pauses function execution
- returns a value,**resumes where it left-off**



### IMPORTANT

**\*\*when a function contains  
yield ,  
it becomes a Generator  
function**





So, how do you use `next()` function with generators

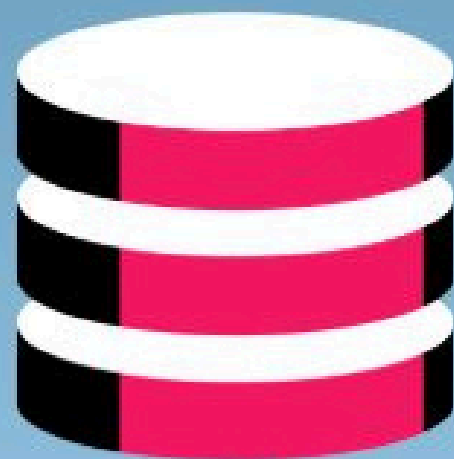
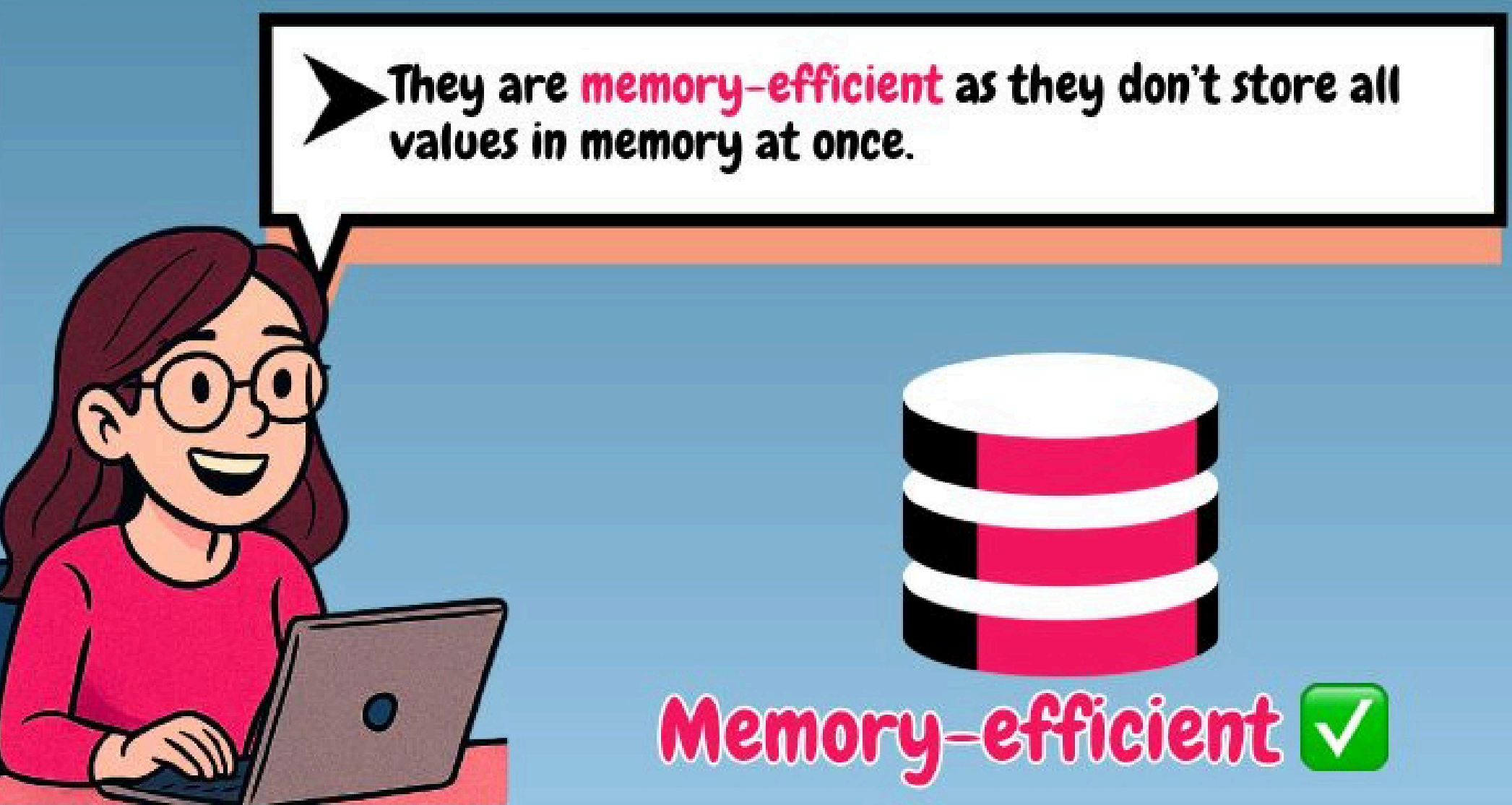
**\*next()**

We use **next()** to retrieve the next value from a **generator**

```
#Using next()  
gen = gen_numbers(5)  
print(next(gen))  
print(next(gen))
```



Why Generator??



Memory-efficient ✓

➤ And that's why , they are useful for **handling large or infinite data streams,**



**Handles Large DataSets** ✓

**Correct!!** ✓




learnwithbhawana





Great.  
What happens when a  
Generator is exhausted?



When a Generator is  
Exhausted ,  
it raises **Stopiteration  
Exception**

## Stopiteration Exception

```
def gen_numbers():  
    yield 1  
  
g = gen_numbers()  
print(next(g)) #Output : 1  
print(next(g)) #StopIteration Error  
  
1
```

```
StopIteration                                     Traceback (most recent call last)  
Cell In[14], line 6  
      4 g = gen_numbers()  
      5 print(next(g)) #Output : 1  
----> 6 print(next(g))  
  
StopIteration:
```

Now,  
How would you handle an error  
if a generator **yields from an  
empty list?**

```
def gen_numbers(list1):  
    for i in list1:  
        yield i  
  
g = gen_numbers([])  
print(next(g)) #Throws Error : StopIteration
```

```
-----  
StopIteration                                     Traceback (most recent call last)  
Cell In[16], line 6  
      3         yield i  
      5 g = gen_numbers([])  
----> 6 print(next(g))  
  
StopIteration:
```

Ummm..

learnwithbhawana

## Option 1 : Check before creating the generator!



```
def gen_numbers(list1):
```

```
    if not list1:  
        print("List is Empty!!")  
        return  
    for i in list1:  
        yield i
```

If condition to ensure the list is not empty

```
g = gen_numbers([])  
print(next(g))
```

List is Empty!!

```
StopIteration  
Cell In[20], line 9  
      6         yield i  
      8 g = gen_numbers([])  
----> 9 print(next(g))
```

Traceback (most recent call last)

StopIteration:

## Option 2 : Catch StopIteration using try-except

```
def gen_numbers(list1):  
    for i in list1:  
        yield i
```

```
try:
```

```
    g = gen_numbers([])  
    print(next(g))
```

```
except StopIteration:  
    print("No values to yield!!")
```

No values to yield!!

Except block to handle StopIteration





Great Job.Last question:  
What do **send()**, **throw()**, and **close()** do in generators?



**send()**  
**throw()**  
**close()**



**send()** sends a value to  
pause yield!



```
def gen():  
    while True:  
        val = yield          # Infinite loop  
                             # Pause & wait  
        print("Got:", val)   # Print value  
  
g = gen()                   # Create generator  
next(g)                     # Start generator  
g.send(42)                  # Send value 42  
  
Got: 42
```



**throw()** raises an exception  
inside generator!

```
def gen():  
    try:  
        yield 1  
    except ValueError:  
        print("ValueError caught!")  
        yield  
  
g = gen()  
next(g)  
g.throw(ValueError)  
  
ValueError caught!
```

*#try block  
#Yield value 1  
#Catch ValueError using except block  
# Print message  
# Resume yielding  
  
# Create generator  
# Start generator  
# Throw error in generator*



**close()** stops the generator  
gracefully!

```
def gen():  
    try:  
        while True:  
            yield  
    finally:  
        print("Closed!")  
  
g = gen()  
next(g)  
g.close()  
  
Closed!
```

*# start try block  
# infinite loop  
# yield control  
# on exit block  
# cleanup message  
  
# create generator  
# start generator  
# close generator*





Well **Bhawana**, Great Job!!  
That's all the question i have on "**Python Generators.**"  
**You've done a great job!!**



**THANK  
YOU!**



Hope my grilling in the Interview  
helped you **prep for Python generators!**

**YOU  
GOT  
THIS!**



learnwithbhawana



learnwithbhawana