Generators in Python

What is a Generator?

- A generator is a special type of function that returns an iterator.
- Unlike normal functions that return all values at once using 'return',
- A generator uses 'yield' to return one value at a time and pauses its execution.
- This makes it memory-efficient, especially for large datasets.

Syntax of a Generator Function

```
def function_name(start, stop, step):
    """
    Example generator function structure:
    - 'yield' returns value to the caller and pauses the function
    - Function execution resumes from the last yield when next() is called
    """
    current = start
    while current < stop:
        yield current    # Value returned to caller
        current += step
```

- Using 'yield' means the function becomes a generator.
- Calling it returns a generator object, not actual values.

How 'yield' Works

- Returns a value to the caller.
- Pauses execution at that point.
- Resumes when next value is requested using next().
- Efficient for memory since values are generated one at a time.

Non-Generator Example

Using normal range() to get numbers 10 to 95 with step 5

```
r = range(10, 100, 5)     # Creates a full list in memory
for val in r:
    print(val)            # Prints all numbers at once
```

Converting Iterable to Iterator

```
tpl = (10, "Rossum", 45.67, True, 2 + 3j)          # Tuple is an iterable
tpliter = iter(tpl)                                 # Convert tuple to iterator

print("Type of tpl =", type(tpl))                   # <class 'tuple'>
print("Type of tpliter =", type(tpliter))           # <class 'tuple_iterator'>

# Using while loop and next() to iterate
while True:
    try:
        print(next(tpliter))
    except StopIteration:                       # StopIteration occurs when all items are exhausted
        break
```

Generator Examples

Simple generator yielding 0 to Val-1

```
def range1(Val):
    i = 0
    while i < Val:
        yield i      # Pause here, return i
        i += 1

# Using next()
g = range1(5)
print(next(g))       # 0
print(next(g))       # 1
print(next(g))       # 2
print(next(g))       # 3
print(next(g))       # 4
# print(next(g))     # StopIteration if uncommented

# Using for-loop automatically handles StopIteration

for val in range1(5):
    print(val)
```

Generator yielding 0 to Val

```
def range2(Val):
    i = 0
    while i <= Val:
        yield i
        i += 1

# Using while + next()
g = range2(10)
while True:
    try:
        print(next(g))
    except StopIteration:
        break

# Using for-loop
for val in range2(10):
    print(val)
```

Generator with start and end

```
def range3(Begin, End):
    while Begin <= End:
        yield Begin
        Begin += 1

# Using next()
g = range3(10, 20)
while True:
    try:
        print(next(g))
    except StopIteration:
        break

# Second generator object
g1 = range3(100, 120)
while True:
    try:
        print(next(g1))
    except StopIteration:
        break

# Using for-loop
for val in range3(10, 20):
    print(val)
```

Generator with start, end, and step

```
def range4(Begin, End, Step):
    while Begin <= End:
        yield Begin
        Begin += Step

g = range4(10, 20, 2)
while True:
    try:
        print(next(g))
    except StopIteration:
        break

g1 = range4(1000, 1050, 10)
while True:
    try:
        print(next(g1))
    except StopIteration:
        break

# Using for-loop
for val in range4(10, 20, 2):
    print(val)
for val in range4(1000, 1050, 10):
    print(val)
```

Generator yielding courses

```
def getcourses():
    yield "PYTHON"
    yield "JAVA"
    yield "DSA"
    yield "DS"

# Using next()
crs = getcourses()
print(next(crs))   # PYTHON
print(next(crs))   # JAVA
print(next(crs))   # DSA
print(next(crs))   # DS

# Using for-loop
for course in getcourses():
    print(course)
```

Notes and Advantages

- Generators use 'yield' to return values one at a time.
- Memory-efficient: values are generated on demand.
- next() resumes from last yield point.
- for-loop handles StopIteration automatically.
- Useful for large datasets or infinite sequences.