Lesson Plan

Generator Functions

PW SKILLS

Topics to be covered:

1. What are Lambda Functions?
2. Working Process of Lambda Functions
3. Lambda Functions Examples
4. Analogy of Lambda Functions
5. Analogy with Lambda Functions Examples

What are Generator Functions?

- Generator functions in Python are a way to create iterators in a more concise and memory-efficient manner.
- These functions allow you to iterate over a potentially large sequence of data without loading the entire sequence into memory.
- The main difference between a regular function and a generator function is that the latter uses the yield keyword to produce a series of values over time, one at a time, rather than returning a single result.

Working Process of Generator Functions:

- **Use of yield:** Instead of using return to send a value back to the caller, a generator function uses **'yield'**. When the generator function encounters a **'yield'** statement, it returns the value specified in the yield and pauses its execution state. The next time the generator is called, it resumes from where it left off.

```
def pw_generator():
    yield 1
    yield 2
    yield 3

# Using the generator
generator = pw_generator()

print(next(generator))  # Output: 1
print(next(generator))  # Output: 2
print(next(generator))  # Output: 3
```

Lazy Evaluation: Generator functions follow the concept of lazy evaluation. This means that the values are generated on-the-fly and only when requested. This is particularly useful for working with large datasets or infinite sequences.

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

# Using the generator for a countdown
countdown_gen = countdown(5)
for number in countdown_gen:
    print(number)
# Output: 5, 4, 3, 2, 1
```

Memory Efficiency: Generators are memory-efficient because they don't store all values in memory at once. Each value is generated and consumed one at a time, making them suitable for working with large datasets or when memory usage is a concern.

```
def fibonacci_generator():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

# Using the generator for Fibonacci sequence
fib_gen = fibonacci_generator()
for _ in range(10):
    print(next(fib_gen))

# Output: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
```

3. Generator Functions Examples:

Example1:

```
def pw_skills_generator():
    yield "Python"
    yield "Web Development"
    yield "Problem Solving"
    yield "Debugging"

# Usage
for skill in pw_skills_generator():
    print(f"Learn: {skill}")
```

```
#output
Learn: Python
Learn: Web Development
Learn: Problem Solving
Learn: Debugging
```

Example2:

```
def courses_generator():
    yield "Python Basics"
    yield "Web Development Fundamentals"
    yield "Data Science Essentials"

# Usage
for course in courses_generator():
    print(f"Enroll in: {course}")

#output
Enroll in: Python Basics
Enroll in: Web Development Fundamentals
Enroll in: Data Science Essentials
```

Example3:

```
def data_science_concepts_generator():
    yield "Data Exploration"
    yield "Machine Learning"
    yield "Data Visualization"
    yield "Statistics Basics"

# Usage
for concept in data_science_concepts_generator():
    print(f"Understand: {concept}")

#output
Understand: Data Exploration
Understand: Machine Learning
Understand: Data Visualization
Understand: Statistics Basics
```

Example4:

```
def data_analytics_tools_generator():
    yield "Pandas"
    yield "NumPy"
    yield "Matplotlib"
    yield "SQL"

# Usage
for tool in data_analytics_tools_generator():
    print(f"Use: {tool}")

#output
Use: Pandas
Use: NumPy
Use: Matplotlib
Use: SQL
```

Example3:

```
def data_science_concepts_generator():
    yield "Data Exploration"
    yield "Machine Learning"
    yield "Data Visualization"
    yield "Statistics Basics"

# Usage
for concept in data_science_concepts_generator():
    print(f"Understand: {concept}")

#output
Understand: Data Exploration
Understand: Machine Learning
Understand: Data Visualization
Understand: Statistics Basics
```