NAME :- CHINMAY SURENDRA BHINTADE

INTERNSHIP DOMAIN :- DATA SCIENCE

COMPANY :- INTRAINZ INNOVATIONS PVT.LTD

MINI PROJECT :- REPORT ON DECORATORS AND GENERATORS IN PYTHON

CONTENT

REPORT ON DECORATORS AND GENERATORS IN PYTHON

- ## INTRODUCTION :-

Python is one of the most popular language and powerful programming languages that can be applied to numerous technologies

The areas like web development , data science , artificial intelligence , machine learning ,etc. gets benefit from this language .There is concept called generators and decorators which is important in functions to be efficient and reusable code. This report discussed the idea of generators and decorators ,functioning , usage, and the advantage it offers when programming in python.

- ## GENERATORS :-

i. INTRODUCTION:-

Generators are python's iterable class that enables us to generate a series of values one at a time while going through them. Unlike lists , generators do not load their full content into memory. Rather they create values as we need them , making them memory – conscious an ideal with massive data sets.

There are two types of generator creation methods:-

1. Generator functions
2. Generator Expressions

Generator Function :- They are declared as regular functions but with the yield keyword in place of return. They return a generator object upon calling.

Generator Expression : These are similar to list comprehensions but use parenthesis () instead of of square bracket [ ].

ii. EXAMPLE:-

1. Generator Using Function:-

```
def number_generator(n):
    for i in range(n):
        yield i

for num in number_generator(5):
    print(num)
```

2. Generator using Expression:

```
squares = (x ** 2 for x in range(5))
for square in squares:
    print(square)
```

iii. ADVANTAGES:-

1. Memory Efficiency : - Generators produce items one at a time which is suitable for large datasets.
2. Lazy valuation:- Values only being produced on demand reduces unnecessary computation.
3. Improved Performance: Generators reduce overhead when dealing with large sequences compared to list-based solutions.

iv. APPLICATIONS:-

1. It can read large databases or files line by line.
2. It can generate infinite sequences for e.g Fibonacci series.
3. The large datasets can be streamed for machine learning models.

- DECORATOR:-

a. INTRODUCTION:-

Decorators are of the powerful features in python that allow changing or enhancing the behaviour of functions or methods without changing their actual code. It helps to make cleaner , more modular code by separation functionality.

Creating and Using Decorators

A decorator is typically a function that takes another function as input, returns the modified function.

b. EXAMPLE:-

```
def decorator_function(original_function):
    def wrapper_function():
        print("Before the function call")
        original_function()
        print("After the function call")
    return wrapper_function

@decorator_function

#using decorator function by @ symbol
def say_hello():
    print("Hello, World!")

say_hello()
```

TYPES OF DECORATORS:-

1. Function Decorator:- applied to standalone functions.
2. Class Decorators:- Applied to classes to modify their behavior.

3. Built in Decorators:- Python provides built in decorators such as @staticmethod,@classmethod, @property.

c. ADVANTAGES:-

1. Code Reusability:- You can implement the same functionality to multiple functions using decorators.

2. Seperation of concerns : It helps keep the logic of functions and their enhancements separate.

3. Dynamic Behavior:- This is because decorators enable you to modify function behavior at runtime.

d. APPLICATIONS :-

1. Decorators are used for debugging and logging.

2. Authentication and authorization in web frameworks like flask and Django.

3. Caching results for the enhancement of performance.

4. Input Validation are the most importance in decorator.

EXAMPLE:-

```
#def logging_decorator(func):
    def wrapper(*args, **kwargs):
    print(f"Calling function: {func.__name__}")
    result = func(*args, **kwargs)
    print(f"Function {func.__name__} executed successfully.")
```

```
        return result
    return wrapper

@logging_decorator
def generate_even_numbers(n):
    for i in range(n):
        if i % 2 == 0:
            yield i

# Using the generator with the decorator
for num in generate_even_numbers(10):
    print(num)
```

- CONCLUSION:-

  Generators and decorators are an essential tool within Python programming, giving both functionality and elegance to the code. Generators enhance the performance and memory efficiency of processing large datasets, which is almost always a requirement in data science.
  Decorators allow for the straightforward, modular extension or modificati on of functions and methods behavior, making them priceless when it comes to building code that can be reused or maintained easily.

  Mastering these tools enhances your coding efficiency while equipping you to solve complex problems in Python, especially in domains like data science, where performance and clarity are critical.