Day 19: Advanced Python – Decorators, Iterators, and Socket Programming

1. Decorators in Python

Introduction to Decorators

A **decorator** is a function that takes another function as input and extends or modifies its behavior **without changing its actual code**. Decorators are often used for **logging, authentication, access control, and timing functions**.

Writing Custom Decorators

Basic Decorator Example

```
# Defining a decorator function
def my_decorator(func):
    def wrapper():
        print("Something before the function runs")
        func()
        print("Something after the function runs")
    return wrapper

# Applying the decorator using '@'
@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```

Output:

```
Something before the function runs
Hello!
Something after the function runs
```

Decorator with Arguments

```
def repeat(n):
    def decorator(func):
        def wrapper(*args, **kwargs):
            for _ in range(n):
                func(*args, **kwargs)
        return wrapper
    return decorator


@repeat(3)
def greet():
    print("Hello!")


greet()
```

Output:

```
Hello!
Hello!
Hello!
```

---

2. Iterators in Python

Iterators vs Iterables

- **Iterable:** An object that can return an iterator (e.g., list, tuple, dict, str). It implements the __iter__() method.
- **Iterator:** An object that **remembers its state** while iterating. It implements both __iter__() and __next__() methods.

Creating a Custom Iterator

```
class MyNumbers:
    def __iter__(self):
        self.num = 1
        return self

    def __next__(self):
        if self.num <= 5:
            val = self.num
            self.num += 1
            return val
        else:
            raise StopIteration

nums = MyNumbers()
my_iter = iter(nums)

for num in my_iter:
    print(num)
```

Output:

```
1

2

3

4

5
```

Built-in Iterators

```
my_list = [10, 20, 30]

iterator = iter(my_list)

print(next(iterator))  # 10

print(next(iterator))  # 20

print(next(iterator))  # 30
```

---

3. Socket Programming in Python

Socket programming allows communication between two computers over a network. Python's socket module enables creating both **client** and **server** applications.

Types of Sockets

1. **TCP (Transmission Control Protocol)** – Reliable, connection-based communication.
2. **UDP (User Datagram Protocol)** – Fast, connectionless communication.

Getting Started with Sockets

Basic Socket Creation

```
import socket


# Creating a socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # TCP Socket

print("Socket created")
```

---

4. Building a TCP Server-Client Program

TCP Server

```
import socket


server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket.bind(("127.0.0.1", 8080))  # Bind to localhost and port 8080

server_socket.listen(1)


print("Waiting for connection...")

conn, addr = server_socket.accept()

print(f"Connected by {addr}")


conn.sendall(b"Hello, Client!")

conn.close()
```

TCP Client

```
import socket


client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

client_socket.connect(("127.0.0.1", 8080))


data = client_socket.recv(1024)

print("Received:", data.decode())


client_socket.close()
```

Output when running the client:

```
Received: Hello, Client!
```

---

5. Understanding UDP Communication

UDP Server

```
import socket


udp_server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
udp_server.bind(("127.0.0.1", 8081))


print("Waiting for data...")
data, addr = udp_server.recvfrom(1024)
print(f"Received {data.decode()} from {addr}")


udp_server.sendto(b"Hello, UDP Client!", addr)
```

UDP Client

```
import socket


udp_client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
udp_client.sendto(b"Hello, Server!", ("127.0.0.1", 8081))


data, addr = udp_client.recvfrom(1024)
print("Received:", data.decode())
```

Output when running the client:

```
Received: Hello, UDP Client!
```

Key Takeaways

1. **Decorators** allow modification of functions without altering their actual implementation.
2. **Iterators** enable controlled iteration over objects and provide memory-efficient traversal.
3. **Socket Programming** enables network communication between client and server using **TCP** and **UDP**.
4. **TCP** is reliable but slower, whereas **UDP** is faster but less reliable.