الاسم : احمد السيد عبده على فرج | سكشن 1

# Chapter 10

**Multiple Choice Questions (MCQs)**

1. Which method is called when entering a context manager block using with?
a) __init__()
b) __enter__()
c) __exit__()
d) __call__()
2. Which keyword is used in Python generators?
a) return
b) yield
c) await
d) break
3. In the Observer pattern, what is the primary responsibility of the Subject?
a) Execute business logic
b) Maintain state and notify observers
c) Inject dependencies
d) Create objects dynamically
4. Which design pattern ensures only one instance of a class exists?
a) Factory
b) Singleton
c) Observer
d) Proxy
5. Which of the following is NOT a benefit of Dependency Injection?
a) Increased flexibility
b) Easier testing
c) Stronger coupling between classes
d) Better modularity    Answer: c) Stronger coupling between classes

**True/False Questions**

The __exit__() method in a context manager is always executed, even if an exception occurs inside the with block. Answer: True
Generators return all values at once like a list. Answer: False
Coroutines can both produce values and receive input using send(). Answer: True
The Factory pattern is used to notify multiple observers when the state of an object changes. Answer: False
Dependency Injection helps reduce coupling between classes. Answer: True

**Short Answer / Conceptual Questions**

What is the difference between a generator and a coroutine in Python? Answer: A generator produces values lazily using yield, while a coroutine can also consume values sent into it using send(). Coroutines are often used for event-driven programming and concurrency.

Explain why the with statement is preferred over manual resource management. Answer: The with statement ensures resources (like files, sockets, or locks) are automatically cleaned up via __exit__(), even if an exception occurs, making code safer and cleaner.
Give a real-world example where the Observer pattern might be applied. Answer: In a stock trading app, multiple UI components (observers) need to be updated whenever stock prices (subject state) change
What problem does the Factory pattern solve? Answer: It abstracts object creation, allowing clients to create objects without depending on their concrete classes.
How does Dependency Injection improve testability of code? Answer: It allows dependencies (like services or databases) to be swapped out with mock objects during testing, making unit tests easier and more isolated.

**Programming Problems**

**Context Manager**

```
import time

class Timer:
    def __enter__(self):
        self.start = time.time()
        return self

    def __exit__(self, *args):
        self.end = time.time()
        print(f"Execution took {self.end - 
self.start:.2f} seconds")

with Timer():
    for i in range(1000000):
        pass

Generator
def even_numbers(n):
    for i in range(2, n + 1, 2):
        yield i

for num in even_numbers(10):
    print(num)
```

**Coroutine**

```
def filter_positive():
    while True:
        num = yield
        if num > 0:
            print(f"Positive number: {num}")

co = filter_positive()
next(co)
co.send(-3)
co.send(5)
co.send(0)
```

**Factory Pattern**

```
class Circle:
    def draw(self):
        return "Drawing a Circle"

class Square:
    def draw(self):
        return "Drawing a Square"

def shape_factory(shape_type):
    if shape_type == "circle":
        return Circle()
    elif shape_type == "square":
        return Square()
    else:
        raise ValueError("Unknown shape")

shape = shape_factory("circle")
print(shape.draw())
```

**Observer Pattern**

```
class Subject:
    def __init__(self):
        self.observers = []

    def attach(self, observer):
        self.observers.append(observer)

    def notify(self, message):
        for obs in self.observers:
            obs.update(message)

class Observer:
    def update(self, message):
        print(f"Received update: {message}")

subject = Subject()
obs1 = Observer()
obs2 = Observer()
subject.attach(obs1)
subject.attach(obs2)
subject.notify("Update available!")
```