

Chapter 10: Patterns for Advanced Python (Page 140)

:Multiple Choice (MCQs)

1. `.enter__()` (Section 10.2) (b)
2. `.yield` (Section 10.3) (b)
3. Maintain state and notify observers (Section 10.4.3.1) (b)
4. Singleton (Section 10.4.1.2) (b)
5. Stronger coupling between classes (False, Dependency Injection (c)
.(coupling reduces

:True/False

1. `.(exit__ guarantees cleanup__)` True
2. `.(Generators yield one at a time/lazy)` False
3. `.(Coroutines use send to receive data)` True
4. `.(Factory is for creation, not notification)` False
- True (DI decouples classes).

```
# =====
# Chapter 10: Patterns
# =====
print("\n--- Chapter 10 Solutions ---")

# 1. Context Manager
class Timer:
    def __enter__(self):
        self.start = time.time()
    def __exit__(self, *args):
        print(f"Execution took {time.time() - self.start:.5f} seconds")

with Timer():
    sum(range(1000000))
```

```

# 2. Generator
def even_numbers(n):
    for i in range(n + 1):
        if i % 2 == 0 and i != 0: yield i

print(f"Evens: {list(even_numbers(10))}")

# 3. Coroutine
def filter_positive():
    while True:
        val = (yield)
        if val > 0: print(f"Positive number: {val}")

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

# 4. Factory Pattern
class ShapeFactory:
    def get_shape(self, type):
        if type == "circle": return "Drawing a Circle"
        if type == "square": return "Drawing a Square"

print(ShapeFactory().get_shape("circle"))

# 5. Observer Pattern
class Subject:
    def __init__(self): self.obs = []
    def attach(self, o): self.obs.append(o)
    def notify(self, m):
        for o in self.obs: o.update(m)

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

s = Subject()
s.attach(Observer())
s.attach(Observer())
s.notify("Update available!")

```