الاسم: احمد وليد عوض عبدالمجيد سيكشن: 1

Chapter 3: Functional Programming (Page 55)

Multiple Choice (MCQs):

9. c) Always returns the same output for the same input (Section 3.2).

10. b) Immutability (Section 3.1).

11. c) map() (Section 3.6.1).

12. c) functools (Section 3.6.3).

13. b) An iterator containing elements that satisfy the condition (Section 3.6.2).

True/False:

14. True (Section 3.1).

15. False (map returns an iterator and does not modify the original list in place).

16. True (Section 3.7 - Closures remember values in enclosing scopes).

17. False (reduce must be imported from functools, as noted in Section 3.6.3).

18. True (Section 3.11).

Short Answer:

20. Differences (Section 3.6):

* map: Applies a function to all items (Transformation).

* filter: Selects items based on a condition (Selection).

* reduce: Cumulatively applies a function to reduce the list to a single value (Aggregation).

الاسم: احمد وليد عوض عبدالمجيد　　　　　　　　سيكشن: 1

Pure Function: *Defined in Section 3.2 as a function that has no side effects and always produces the same output for the same input. Example: def add(a, b): return a + b.*

```python
# ==========================================
# Chapter 3: Functional Programming
# ==========================================
print("\n--- Chapter 3 Solutions ---")

# 1. Remove Vowels
def remove_vowels(text):
    return "".join([c for c in text if c.lower() not in "aeiou"])

print(f"Remove Vowels: {remove_vowels('Hello World')}")

# 2. Map and Filter
nums = [1, 2, 3, 4, 5]
sq_odd = list(map(lambda x: x**2, filter(lambda x: x % 2 != 0, nums)))
print(f"Squares of odds: {sq_odd}")

# 3. Fibonacci with lru_cache
@functools.lru_cache(maxsize=None)
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)

print(f"Fibonacci(10): {fib(10)}")

# 4. Closure make_adder
def make_adder(n):
    return lambda x: x + n

add5 = make_adder(5)
print(f"make_adder(5)(10): {add5(10)}")

# 5. Higher Order Function apply_twice
def apply_twice(func, value):
    return func(func(value))

print(f"apply_twice: {apply_twice(lambda x: x + 1, 5)}")

# 6. ETL Pipeline
def etl_pipeline(texts):
    stopwords = {"the", "a", "is"}
```

الاسم: احمد وليد عوض عبدالمجيد سيكشن: 1

```python
    words = [w.lower() for t in texts for w in t.split()]
    filtered = [w for w in words if w not in stopwords]
    return {w: filtered.count(w) for w in set(filtered)}

print(f"ETL: {etl_pipeline(['Python is great', 'The Python code'])}")

# 7. Custom Reduce
def my_reduce(func, iterable, initializer=None):
    it = iter(iterable)
    value = next(it) if initializer is None else initializer
    for element in it:
        value = func(value, element)
    return value

print(f"Custom Reduce (Sum): {my_reduce(lambda x, y: x+y, [1, 2, 3, 4])}")

# 8. Decorator log_call
def log_call(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        print(f"Calling {func.__name__}...")
        result = func(*args, **kwargs)
        print(f"{func.__name__} finished.")
        return result
    return wrapper

@log_call
def test_log(): pass
test_log()
```