Python Practice Challenges - Chapter 2

Lists and Functions

Instructions

Complete each challenge by writing Python functions. Test your solutions with the provided test cases and create additional tests.

Challenge 1: List Basics

Create functions that demonstrate fundamental list operations.

Your Task:

```
# Create these functions:
# 1. create_number_list(start, end) - returns list of numbers from start to
end
# 2. get_list_info(lst) - returns dict with length, first, last, and sum
# 3. reverse_list(lst) - returns new list in reverse order
# 4. merge_lists(list1, list2) - returns combined list without duplicates
```

Test Cases:

```
numbers = create_number_list(1, 5)
print(numbers)                    # [1, 2, 3, 4, 5]

info = get_list_info([10, 20, 30, 40])
print(info)                       # {'length': 4, 'first': 10, 'last': 40,
'sum': 100}

reversed_nums = reverse_list([1, 2, 3, 4])
print(reversed_nums)              # [4, 3, 2, 1]

merged = merge_lists([1, 2, 3], [3, 4, 5])
print(merged)                     # [1, 2, 3, 4, 5]
```

Challenge 2: Number Analysis Functions

Create functions that analyze lists of numbers.

Your Task:

```
# Create these functions:
```

```
# 1. find_extremes(numbers) - returns tuple (min, max, range)
# 2. count_by_parity(numbers) - returns dict with counts of odd/even numbers
# 3. above_average(numbers) - returns list of numbers above the average
# 4. statistical_summary(numbers) - returns comprehensive stats dict
```

Test Cases:

```
data = [12, 5, 8, 15, 3, 9, 20, 7]

extremes = find_extremes(data)
print(extremes)                     # (3, 20, 17)

parity = count_by_parity(data)
print(parity)                       # {'odd': 4, 'even': 4}

above_avg = above_average(data)
print(above_avg)                    # [12, 15, 20] (average is 9.875)

summary = statistical_summary(data)
print(summary)
# {'count': 8, 'sum': 79, 'average': 9.875, 'min': 3, 'max': 20, 'range': 17}
```

Challenge 3: String List Processing

Work with lists of strings and text processing.

Your Task:

```
# Create these functions:
# 1. filter_by_length(words, min_length, max_length) - filter words by length
# 2. find_words_with_letter(words, letter) - find words containing specific
letter
# 3. sort_by_length(words) - sort words by length, then alphabetically
# 4. create_acronym(words) - create acronym from first letters
```

Test Cases:

```
words = ["apple", "banana", "cat", "dog", "elephant", "fox", "grape"]

filtered = filter_by_length(words, 3, 5)
print(filtered)                     # ["apple", "cat", "dog", "fox", "grape"]

with_a = find_words_with_letter(words, "a")
print(with_a)                       # ["apple", "banana", "cat", "elephant",
"grape"]

sorted_words = sort_by_length(words)
print(sorted_words)                 # ["cat", "dog", "fox", "apple", "grape",
"banana", "elephant"]

acronym = create_acronym(["Python", "Is", "Great"])
print(acronym)                      # "PIG"
```

Challenge 4: Advanced List Operations

Implement more complex list manipulation functions.

Your Task:

```
# Create these functions:
# 1. chunk_list(lst, chunk_size) - split list into chunks of given size
# 2. flatten_list(nested_list) - flatten a list of lists
# 3. rotate_list(lst, positions) - rotate list by given positions
# 4. find_common_elements(list1, list2) - find elements present in both lists
```

Test Cases:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
chunks = chunk_list(numbers, 3)
print(chunks)                     # [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

nested = [[1, 2], [3, 4], [5, 6]]
flattened = flatten_list(nested)
print(flattened)                  # [1, 2, 3, 4, 5, 6]

rotated = rotate_list([1, 2, 3, 4, 5], 2)
print(rotated)                    # [4, 5, 1, 2, 3]

common = find_common_elements([1, 2, 3, 4], [3, 4, 5, 6])
print(common)                     # [3, 4]
```

Challenge 5: Shopping Cart System

Build a comprehensive shopping cart system using lists and functions.

Your Task:

```
# Item structure: {"name": str, "price": float, "quantity": int}
# Create these functions:
# 1. create_item(name, price, quantity) - create item dictionary
# 2. add_to_cart(cart, item) - add item to cart (merge if exists)
# 3. remove_from_cart(cart, item_name) - remove item completely
# 4. update_quantity(cart, item_name, new_quantity) - update item quantity
# 5. calculate_total(cart) - calculate total price
# 6. apply_discount(cart, discount_percent) - apply discount to total
# 7. generate_receipt(cart) - create formatted receipt string
```

Test Cases:

```
cart = []
```

```
# Add items
apple = create_item("Apple", 1.50, 3)
bread = create_item("Bread", 2.99, 1)
milk = create_item("Milk", 3.49, 2)

add_to_cart(cart, apple)
add_to_cart(cart, bread)
add_to_cart(cart, milk)

print(f"Total: ${calculate_total(cart):.2f}")  # Total: $11.47

update_quantity(cart, "Apple", 5)
remove_from_cart(cart, "Bread")

discounted_total = apply_discount(cart, 10)    # 10% discount
print(f"After discount: ${discounted_total:.2f}")

receipt = generate_receipt(cart)
print(receipt)
```

Expected Receipt Format:

```
========== RECEIPT ==========
Apple         x5    $7.50
Milk          x2    $6.98
                   ------
Subtotal:          $14.48
Discount (10%):    -$1.45
TOTAL:             $13.03
============================
```

Challenge 6: Grade Management System

Create a complete grade management system with advanced features.

Your Task:

```
# Student structure: {"name": str, "grades": [float], "assignments": [str]}
# Create these functions:
# 1. create_student(name) - create student with empty grades
# 2. add_grade(student, assignment, grade) - add grade for assignment
# 3. calculate_gpa(grades) - convert average to 4.0 scale
# 4. get_letter_grade(average) - convert average to letter grade
# 5. generate_transcript(student) - create detailed transcript
# 6. class_ranking(students) - rank students by GPA
# 7. find_top_performers(students, percentage) - find top % of students
```

Test Cases:

```
# Create students
alice = create_student("Alice Johnson")
```

```
bob = create_student("Bob Smith")

# Add grades
add_grade(alice, "Math Test 1", 92)
add_grade(alice, "English Essay", 88)
add_grade(alice, "Science Quiz", 95)

add_grade(bob, "Math Test 1", 78)
add_grade(bob, "English Essay", 82)
add_grade(bob, "Science Quiz", 85)

students = [alice, bob]

# Generate reports
alice_transcript = generate_transcript(alice)
rankings = class_ranking(students)
top_students = find_top_performers(students, 50)  # Top 50%

print(alice_transcript)
print(f"Rankings: {rankings}")
print(f"Top performers: {[s['name'] for s in top_students]}")
```

Challenge 7: Data Analysis Tools

Create functions for analyzing and visualizing data patterns.

Your Task:

```
# Create these functions:
# 1. frequency_counter(items) - count frequency of each item
# 2. find_mode(numbers) - find most frequent number(s)
# 3. quartiles(numbers) - calculate Q1, Q2 (median), Q3
# 4. outliers(numbers) - find outliers using IQR method
# 5. normalize_data(numbers) - normalize to 0-1 range
# 6. correlation(list1, list2) - simple correlation coefficient
```

Test Cases:

```
data = [1, 2, 2, 3, 3, 3, 4, 4, 5, 100]  # 100 is outlier

freq = frequency_counter(data)
print(freq)                        # {1: 1, 2: 2, 3: 3, 4: 2, 5: 1, 100: 1}

mode = find_mode(data)
print(mode)                        # [3]

q1, q2, q3 = quartiles(data)
print(f"Q1: {q1}, Q2: {q2}, Q3: {q3}")

outlier_values = outliers(data)
print(outlier_values)              # [100]

normalized = normalize_data([10, 20, 30, 40, 50])
```

```
print(normalized)                  # [0.0, 0.25, 0.5, 0.75, 1.0]
```

Challenge 8: Mini Project - Inventory Management

Create a complete inventory management system.

Your Task: Create an inventory system with the following functions:

```
# Product structure: {"id": int, "name": str, "price": float, "stock": int,
"category": str}
# Required functions:
# 1. create_inventory() - return empty inventory list
# 2. add_product(inventory, name, price, stock, category) - add product with
auto ID
# 3. find_product_by_id(inventory, product_id) - find product by ID
# 4. find_products_by_category(inventory, category) - filter by category
# 5. update_stock(inventory, product_id, new_stock) - update stock level
# 6. low_stock_alert(inventory, threshold) - find products below threshold
# 7. calculate_inventory_value(inventory) - total value of all products
# 8. generate_inventory_report(inventory) - comprehensive report
# 9. search_products(inventory, search_term) - search by name
# 10. sort_by_price(inventory, ascending=True) - sort products by price
```

Complete Test System:

```
# Initialize system
inventory = create_inventory()

# Add products
add_product(inventory, "Laptop", 999.99, 10, "Electronics")
add_product(inventory, "Mouse", 29.99, 50, "Electronics")
add_product(inventory, "Desk", 199.99, 5, "Furniture")
add_product(inventory, "Chair", 149.99, 8, "Furniture")
add_product(inventory, "Keyboard", 79.99, 3, "Electronics")

# Test operations
print("=== Inventory Management System ===")
print(f"Total inventory value: ${calculate_inventory_value(inventory):.2f}")

low_stock = low_stock_alert(inventory, 5)
print(f"Low stock items: {[p['name'] for p in low_stock]}")

electronics = find_products_by_category(inventory, "Electronics")
print(f"Electronics count: {len(electronics)}")

sorted_by_price = sort_by_price(inventory, ascending=False)
print(f"Most expensive: {sorted_by_price[0]['name']}")

search_results = search_products(inventory, "Chair")
print(f"Search results: {[p['name'] for p in search_results]}")

report = generate_inventory_report(inventory)
print(report)
```

Expected Report Format:

```
============ INVENTORY REPORT ============
Total Products: 5
Total Categories: 2
Total Value: $1,459.92

By Category:
  Electronics: 3 items, $1,109.94 value
  Furniture: 2 items, $349.98 value

Low Stock Alerts:
  - Keyboard (3 units remaining)
  - Desk (5 units remaining)

Top 3 Most Valuable Items:
  1. Laptop - $999.99 (10 in stock)
  2. Desk - $199.99 (5 in stock)
  3. Chair - $149.99 (8 in stock)
=========================================
```

Solutions Guide

Challenge 1 Solutions:

```
def create_number_list(start, end):
    return list(range(start, end + 1))

def get_list_info(lst):
    if not lst:
        return {'length': 0, 'first': None, 'last': None, 'sum': 0}
    return {
        'length': len(lst),
        'first': lst[0],
        'last': lst[-1],
        'sum': sum(lst)
    }

def reverse_list(lst):
    return lst[::-1]

def merge_lists(list1, list2):
    # Convert to sets to remove duplicates, then back to sorted list
    combined = set(list1 + list2)
    return sorted(list(combined))
```

Challenge 2 Solutions:

```
def find_extremes(numbers):
    if not numbers:
        return None
    min_val = min(numbers)
    max_val = max(numbers)
    return (min_val, max_val, max_val - min_val)

def count_by_parity(numbers):
    odd_count = sum(1 for num in numbers if num % 2 == 1)
    even_count = len(numbers) - odd_count
    return {'odd': odd_count, 'even': even_count}

def above_average(numbers):
    if not numbers:
        return []
    avg = sum(numbers) / len(numbers)
    return [num for num in numbers if num > avg]

def statistical_summary(numbers):
    if not numbers:
        return {}

    total = sum(numbers)
    count = len(numbers)
    average = total / count
    min_val = min(numbers)
    max_val = max(numbers)

    return {
        'count': count,
        'sum': total,
        'average': average,
        'min': min_val,
        'max': max_val,
        'range': max_val - min_val
    }
```

Challenge 3 Solutions:

```
def filter_by_length(words, min_length, max_length):
    return [word for word in words if min_length <= len(word) <= max_length]

def find_words_with_letter(words, letter):
    return [word for word in words if letter.lower() in word.lower()]

def sort_by_length(words):
    return sorted(words, key=lambda word: (len(word), word.lower()))

def create_acronym(words):
    return ''.join(word[0].upper() for word in words if word)
```

Challenge 5 Partial Solution:

```
def create_item(name, price, quantity):
    return {"name": name, "price": price, "quantity": quantity}

def add_to_cart(cart, item):
    # Check if item already exists
    for existing_item in cart:
        if existing_item["name"] == item["name"]:
            existing_item["quantity"] += item["quantity"]
            return
    # If not found, add new item
    cart.append(item.copy())

def calculate_total(cart):
    return sum(item["price"] * item["quantity"] for item in cart)

def apply_discount(cart, discount_percent):
    total = calculate_total(cart)
    discount_amount = total * (discount_percent / 100)
    return total - discount_amount

def generate_receipt(cart):
    receipt = "========== RECEIPT ==========\n"
    subtotal = 0

    for item in cart:
        line_total = item["price"] * item["quantity"]
        subtotal += line_total
        receipt += f"{item['name']:<12} x{item['quantity']:<4}
${line_total:.2f}\n"

    receipt += "                   ------\n"
    receipt += f"Subtotal:          ${subtotal:.2f}\n"
    receipt += "============================"
    return receipt
```

Additional Practice Exercises

Exercise A: List Comprehension Mastery

```
# Practice these list comprehension patterns:

# 1. Square all even numbers in a list
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squared_evens = [x**2 for x in numbers if x % 2 == 0]

# 2. Create a list of tuples (number, is_even)
number_info = [(x, x % 2 == 0) for x in numbers]

# 3. Flatten a matrix
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flattened = [num for row in matrix for num in row]

# 4. Filter and transform strings
words = ["hello", "world", "python", "programming"]
capitalized_long = [word.upper() for word in words if len(word) > 5]
```

Exercise B: Function Decorators (Advanced)

```
def timer_decorator(func):
    """A simple decorator to time function execution."""
    import time
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"{func.__name__} took {end_time - start_time:.4f} seconds")
        return result
    return wrapper

@timer_decorator
def slow_function(n):
    """A function that takes some time to complete."""
    total = 0
    for i in range(n):
        total += i
    return total

# Usage
result = slow_function(1000000)
```

Exercise C: Error Handling in Functions

```
def safe_divide(a, b):
    """Safely divide two numbers with error handling."""
    try:
        result = a / b
        return {"success": True, "result": result}
    except ZeroDivisionError:
        return {"success": False, "error": "Cannot divide by zero"}
    except TypeError:
        return {"success": False, "error": "Invalid input types"}

def safe_list_access(lst, index):
    """Safely access list element with error handling."""
    try:
        return {"success": True, "value": lst[index]}
    except IndexError:
        return {"success": False, "error": f"Index {index} out of range"}
    except TypeError:
        return {"success": False, "error": "Invalid list or index type"}

# Usage
print(safe_divide(10, 2))      # {"success": True, "result": 5.0}
print(safe_divide(10, 0))      # {"success": False, "error": "Cannot divide 
by zero"}
print(safe_list_access([1, 2, 3], 1))  # {"success": True, "value": 2}
print(safe_list_access([1, 2, 3], 5))  # {"success": False, "error": "Index 5 
out of range"}
```

Final Project: Complete Task Management System

Your Ultimate Challenge: Create a comprehensive task management system that demonstrates all the concepts from this chapter.

Requirements:

```
# Task structure: {"id": int, "title": str, "description": str, "priority":
str, "status": str, "due_date": str}
# Priority: "Low", "Medium", "High"
# Status: "Pending", "In Progress", "Completed"

# Required Functions:
# 1. create_task_manager() - initialize empty task list
# 2. add_task(manager, title, description, priority, due_date) - add new task
# 3. mark_complete(manager, task_id) - mark task as completed
# 4. filter_by_status(manager, status) - filter tasks by status
# 5. filter_by_priority(manager, priority) - filter tasks by priority
# 6. sort_by_due_date(manager) - sort tasks by due date
# 7. search_tasks(manager, keyword) - search in title/description
# 8. get_task_statistics(manager) - return summary statistics
# 9. generate_task_report(manager) - create formatted report
# 10. export_tasks_csv(manager, filename) - export to CSV format

# Bonus: Add recurring tasks, task dependencies, time tracking
```

This final project should demonstrate:

- List manipulation and filtering
- Function design and organization
- String processing and formatting
- Data analysis and reporting
- Error handling and validation
- Code organization and documentation

Good luck with your Python programming journey!