Python Problem-Solving Workshop

DAY 1: Python Basics & File Handling

Python Basics - Part A

Exercise 1.1: Temperature Converter

- **Learning Objective:** Master basic input/output, arithmetic operations, and function creation
- **Difficulty:** Beginner
- **Time:** 20 minutes

Problem Description:
Create a program that converts temperatures between Celsius, Fahrenheit, and Kelvin. The user should be able to choose the conversion type and input a value.

Expected Output:

```
Temperature Converter
1. Celsius to Fahrenheit
2. Fahrenheit to Celsius
3. Celsius to Kelvin
4. Kelvin to Celsius
Choose conversion (1-4): 1
Enter temperature: 25
25.0°C = 77.0°F
```

Hints:

- F = C × 9/5 + 32
- K = C + 273.15
- Use functions for each conversion
- Validate user input

Exercise 1.2: List Manipulator

- **Learning Objective:** Work with lists, loops, and list methods
- **Difficulty:** Beginner

- **Time:** 25 minutes

Problem Description:
Write a program that takes a list of numbers and provides the following operations:

1. Find the sum and average
2. Find the maximum and minimum
3. Remove duplicates
4. Sort in ascending and descending order
5. Filter even and odd numbers

Expected Output:

```
Enter numbers separated by spaces: 5 3 8 3 9 1 8 2
Original list: [5, 3, 8, 3, 9, 1, 8, 2]
Sum: 39
Average: 4.875
Max: 9, Min: 1
Without duplicates: [5, 3, 8, 9, 1, 2]
Sorted (asc): [1, 2, 3, 5, 8, 9]
Even numbers: [8, 2]
Odd numbers: [5, 3, 9, 1]
```

Hints:

- Use built-in functions: sum(), max(), min()
- Use set() to remove duplicates
- Use list comprehension for filtering

Exercise 1.3: String Pattern Analyzer

- **Learning Objective:** String manipulation, dictionaries, and character counting
- **Difficulty:** Beginner
- **Time:** 25 minutes

Problem Description:
Create a program that analyzes a given text string and provides:

1. Character frequency count (case-insensitive)
2. Word count
3. Most common character (excluding spaces)
4. Palindrome check
5. Reverse the string

Expected Output:

```
Enter text: Python Programming
Total characters (with spaces): 18
Total characters (without spaces): 16
Total words: 2
Character frequency: {'p': 2, 'y': 1, 't': 1, 'h': 1, 'o': 2, 'n': 2, 'r':
2, 'g': 2, 'a': 1, 'm': 2, 'i': 1}
Most common character: p, r, o, n, g, m (appears 2 times)
Is palindrome: No
Reversed: gnimmargorP nohtyP
```

Hints:

- Use .lower() for case-insensitive comparison
- Use dictionaries to count frequency
- Use .split() for word count

Python Basics - Part B

Exercise 1.4: Number Guessing Game

- **Learning Objective:** Control flow, loops, conditionals, and random module
- **Difficulty:** Beginner
- **Time:** 30 minutes

Problem Description:
Create a number guessing game where:

- The program generates a random number between 1 and 100
- The user has 7 attempts to guess
- After each guess, provide hints (too high/too low)
- Track the number of attempts
- Ask if the user wants to play again

Expected Output:

```
Welcome to Number Guessing Game!
I'm thinking of a number between 1 and 100.
You have 7 attempts.

Attempt 1 - Your guess: 50
Too low! Try again.
```

```
Attempt 2 - Your guess: 75
Too high! Try again.

Attempt 3 - Your guess: 63
Congratulations! You guessed it in 3 attempts!

Play again? (yes/no): no
Thanks for playing!
```

Hints:

- Use `import random` and `random.randint(1, 100)`
- Use a while loop with a counter
- Implement input validation

Exercise 1.5: Shopping Cart System

- **Learning Objective:** Dictionaries, nested data structures, and functions
- **Difficulty:** Intermediate
- **Time:** 40 minutes

Problem Description:
Build a shopping cart system with the following features:

1. Add items (name, price, quantity)
2. Remove items
3. Update quantity
4. Calculate total price
5. Apply discount code (10% off if total > $100)
6. Display cart summary

Expected Output:

```
Shopping Cart Menu:
1. Add item
2. Remove item
3. Update quantity
4. View cart
5. Checkout
6. Exit

Choice: 1
```

```
Item name: Laptop
Price: 800
Quantity: 1
Item added!

Choice: 4
--- Cart Summary ---
Item         Qty    Price     Total
Laptop       1      $800      $800
Mouse        2      $25       $50
---------------------------------
Subtotal: $850
Discount: $85 (10% off)
Total: $765
```

Hints:

- Use a dictionary with item names as keys
- Store price and quantity as nested dictionary
- Create separate functions for each operation

File Handling

Exercise 1.6: Contact Book Manager

- **Learning Objective:** File reading/writing, CSV handling, data persistence
- **Difficulty:** Beginner
- **Time:** 35 minutes

Problem Description:
Create a contact book manager that stores contacts in a text file. Features:

1. Add new contact (name, phone, email)
2. View all contacts
3. Search contact by name
4. Delete contact
5. Update contact information

Expected Output:

```
Contact Book Manager
1. Add Contact
2. View All Contacts
```

```
3. Search Contact
4. Delete Contact
5. Update Contact
6. Exit

Choice: 1
Name: John Doe
Phone: 555-1234
Email: john@example.com
Contact saved!

Choice: 2
--- All Contacts ---
1. John Doe | 555-1234 | john@example.com
2. Jane Smith | 555-5678 | jane@example.com
```

Hints:

- Use `open()` with modes 'r', 'w', 'a'
- Store each contact on a new line with delimiter (e.g., `|`)
- Use `with` statement for file handling

Exercise 1.7: Log File Analyzer

- **Learning Objective:** File reading, string parsing, data analysis
- **Difficulty:** Intermediate
- **Time:** 40 minutes

Problem Description:
Create a program that analyzes a log file and provides statistics. Given a log file format:

```
2024-01-15 10:23:45 [INFO] User logged in
2024-01-15 10:24:12 [ERROR] Database connection failed
2024-01-15 10:25:03 [WARNING] High memory usage
```

Provide:

1. Total number of log entries
2. Count by log level (INFO, WARNING, ERROR)
3. List all ERROR messages
4. Find the most common log level
5. Export filtered logs (e.g., only ERRORs) to a new file

Expected Output:

```
Log File Analysis
-----------------
Total entries: 150
INFO: 120
WARNING: 20
ERROR: 10

Most common level: INFO

ERROR messages:
1. 2024-01-15 10:24:12 - Database connection failed
2. 2024-01-15 11:30:45 - File not found

Filtered log exported to 'errors.log'
```

Hints:

- Read file line by line
- Use string methods like .split() and .startswith()
- Use regular expressions for pattern matching (optional)

Exercise 1.8: CSV Data Processor

- **Learning Objective:** CSV module, data transformation, file I/O
- **Difficulty:** Intermediate
- **Time:** 45 minutes

Problem Description:
Process a CSV file containing student grades and generate a report:

Input CSV (students.csv):

```
Name,Math,Science,English
Alice,85,90,88
Bob,78,82,75
Charlie,92,88,94
```

Tasks:

1. Read the CSV file
2. Calculate average grade for each student
3. Find the class average for each subject

4. Identify top performer
5. Generate a report and save to a new CSV file with averages included

Expected Output:

```
Student Report Generated
------------------------
Alice - Average: 87.67
Bob - Average: 78.33
Charlie - Average: 91.33

Class Averages:
Math: 85.0
Science: 86.67
English: 85.67

Top Performer: Charlie (91.33)

Report saved to 'student_report.csv'
```

Hints:

- Use `import csv` module
- Use `csv.DictReader` for easy column access
- Use `csv.DictWriter` for writing with headers

DAY 2: OOP, Iterators, Generators & Decorators

Object-Oriented Programming - Part A

Exercise 2.1: Bank Account Class

- **Learning Objective:** Class creation, attributes, methods, encapsulation
- **Difficulty:** Beginner
- **Time:** 25 minutes

Problem Description:
Create a `BankAccount` class with:

- Attributes: account_number, holder_name, balance
- Methods: deposit(), withdraw(), check_balance(), get_account_info()
- Prevent withdrawal if insufficient funds

- Keep track of transaction history

Expected Output:

```
account = BankAccount("12345", "Alice", 1000)
account.deposit(500)
# Output: Deposited $500. New balance: $1500

account.withdraw(200)
# Output: Withdrawn $200. New balance: $1300

account.withdraw(2000)
# Output: Insufficient funds. Current balance: $1300

print(account.get_account_info())
# Output: Account: 12345 | Holder: Alice | Balance: $1300
```

Hints:

- Use `__init__` constructor
- Use instance variables (self.attribute)
- Add validation in withdraw method

Exercise 2.2: Library Management System

- **Learning Objective:** Multiple classes, relationships, class methods
- **Difficulty:** Intermediate
- **Time:** 45 minutes

Problem Description:
Create a library management system with two classes:

Book class:

- Attributes: title, author, isbn, is_available
- Methods: borrow(), return_book()

Library class:

- Attributes: name, books (list)
- Methods: add_book(), remove_book(), search_book(), display_available_books(), borrow_book(), return_book()

Expected Output:

```
library = Library("City Library")
book1 = Book("Python Crash Course", "Eric Matthes", "978-1593279288")
library.add_book(book1)

library.display_available_books()
# Output: Available Books:
#          1. Python Crash Course by Eric Matthes (ISBN: 978-1593279288)

library.borrow_book("978-1593279288")
# Output: Book 'Python Crash Course' borrowed successfully

library.display_available_books()
# Output: No books currently available

library.return_book("978-1593279288")
# Output: Book 'Python Crash Course' returned successfully
```

Hints:

- Book class manages individual book status
- Library class manages collection of books
- Use list to store multiple Book objects

Object-Oriented Programming - Part B

Exercise 2.3: Shape Hierarchy with Inheritance

- **Learning Objective:** Inheritance, method overriding, polymorphism
- **Difficulty:** Intermediate
- **Time:** 35 minutes

Problem Description:
Create a shape hierarchy:

Base class Shape:

- Attributes: color, name
- Methods: area() (abstract), perimeter() (abstract), display_info()

Derived classes:

- Circle (radius)
- Rectangle (length, width)

- Triangle (base, height, sides for perimeter)

Each class should implement area() and perimeter() methods.

Expected Output:

```
circle = Circle("Red", 5)
rectangle = Rectangle("Blue", 4, 6)
triangle = Triangle("Green", 3, 4, 3, 4, 5)

shapes = [circle, rectangle, triangle]
for shape in shapes:
    shape.display_info()

# Output:
# Shape: Circle | Color: Red | Area: 78.54 | Perimeter: 31.42
# Shape: Rectangle | Color: Blue | Area: 24 | Perimeter: 20
# Shape: Triangle | Color: Green | Area: 6.0 | Perimeter: 12
```

Hints:

- Use inheritance with `class Derived(Base):`
- Override methods in derived classes
- Use `super()` to call parent class constructor
- Import math module for π

Exercise 2.4: Employee Management System

- **Learning Objective:** Complex inheritance, special methods, class variables
- **Difficulty:** Intermediate
- **Time:** 40 minutes

Problem Description:
Create an employee management system:

Base class Employee:

- Attributes: name, employee_id, base_salary
- Class variable: company_name, total_employees
- Methods: calculate_salary() (abstract), get_info(), `__str__`, `__repr__`

Derived classes:

- FullTimeEmployee (base_salary, benefits)

- PartTimeEmployee (hourly_rate, hours_worked)
- Contractor (project_fee, projects_completed)

Expected Output:

```
emp1 = FullTimeEmployee("Alice", "E001", 50000, 10000)
emp2 = PartTimeEmployee("Bob", "E002", 25, 80)
emp3 = Contractor("Charlie", "E003", 5000, 3)

print(emp1)
# Output: FullTimeEmployee(Alice, E001) - Salary: $60000

print(Employee.total_employees)
# Output: 3

employees = [emp1, emp2, emp3]
total_payroll = sum(emp.calculate_salary() for emp in employees)
print(f"Total Payroll: ${total_payroll}")
# Output: Total Payroll: $77000
```

Hints:

- Use class variables for shared data
- Implement `__str__` for readable output
- Implement `__repr__` for debugging
- Each derived class calculates salary differently

Iterators, Generators & Decorators

Exercise 2.5: Custom Iterator

- **Learning Objective:** Implement `__iter__` and `__next__` methods
- **Difficulty:** Intermediate
- **Time:** 30 minutes

Problem Description:
Create a custom iterator class `FibonacciIterator` that generates Fibonacci numbers up to a specified count.

Additionally, create a `RangeIterator` that mimics Python's built-in range() function.

Expected Output:

```
fib = FibonacciIterator(10)
for num in fib:
    print(num, end=" ")
# Output: 0 1 1 2 3 5 8 13 21 34

print("\n")

custom_range = RangeIterator(5, 15, 2)
for num in custom_range:
    print(num, end=" ")
# Output: 5 7 9 11 13
```

Hints:

- Implement `__iter__` to return self
- Implement `__next__` to return next value
- Raise `StopIteration` when done
- Maintain internal state

Exercise 2.6: Generator Functions

- **Learning Objective:** Create and use generator functions with yield
- **Difficulty:** Intermediate
- **Time:** 35 minutes

Problem Description:
Create the following generators:

1. **Prime Generator:** Generate prime numbers up to n
2. **File Reader:** Read large files line by line (memory efficient)
3. **Infinite Sequence:** Generate infinite arithmetic or geometric sequences
4. **Batch Generator:** Split a list into batches of specified size

Expected Output:

```
# Prime Generator
for prime in prime_generator(20):
    print(prime, end=" ")
# Output: 2 3 5 7 11 13 17 19

# Batch Generator
data = list(range(1, 11))
```

```
for batch in batch_generator(data, 3):
    print(batch)
# Output: [1, 2, 3]
#         [4, 5, 6]
#         [7, 8, 9]
#         [10]

# Infinite Arithmetic Sequence
seq = arithmetic_sequence(start=5, step=3)
for i, num in enumerate(seq):
    if i >= 5:
        break
    print(num, end=" ")
# Output: 5 8 11 14 17
```

Hints:

- Use `yield` instead of `return`
- Generators are memory efficient
- Use `next()` to manually advance
- Can loop over generators with for loop

Exercise 2.7: Function Decorators

- **Learning Objective:** Create and apply decorators, understand wrapper functions
- **Difficulty:** Intermediate
- **Time:** 40 minutes

Problem Description:
Create the following decorators:

1. **@timer:** Measure execution time of a function
2. **@logger:** Log function calls with arguments and return values
3. **@validate_args:** Validate function arguments (e.g., type checking)
4. **@retry:** Retry a function if it fails (with max attempts)
5. **@cache:** Memoization decorator for expensive functions

Expected Output:

```
@timer
def slow_function():
    time.sleep(2)
    return "Done"
```

```
result = slow_function()
# Output: Function 'slow_function' took 2.003 seconds

@logger
def add(a, b):
    return a + b

result = add(5, 3)
# Output: Calling add with args=(5, 3), kwargs={}
#         add returned 8

@cache
def fibonacci(n):
    if n < 2:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(35))  # Fast due to caching
# Output: 9227465

@retry(max_attempts=3)
def unstable_api_call():
    # Simulated API that might fail
    if random.random() < 0.7:
        raise Exception("API Error")
    return "Success"

result = unstable_api_call()
# Output: Attempt 1 failed: API Error
#         Attempt 2 succeeded
#         Success
```

Hints:

- Use `functools.wraps` to preserve function metadata
- Decorator structure: function that returns wrapper function
- Access original function args with *args, **kwargs
- Use closures to maintain state (for cache decorator)

Advanced Integration

Exercise 2.8: Data Pipeline with Generators and Decorators

- **Learning Objective:** Combine OOP, generators, and decorators
- **Difficulty:** Intermediate
- **Time:** 60 minutes

Problem Description:
Create a data processing pipeline that:

1. **DataSource class:** Reads data from a file using a generator
2. **DataProcessor class:** Applies transformations (filter, map, reduce)
3. Use decorators to:
    - Log each processing step
    - Time each operation
    - Cache expensive transformations
4. Process a CSV file with student data:
    - Filter by passing grade (>= 60)
    - Calculate grade letter (A, B, C, D, F)
    - Generate statistics

Expected Output:

```python
pipeline = DataPipeline("students.csv")

# Use generator to read
for student in pipeline.read_data():
    print(student)

# Apply transformations
pipeline.filter_by_grade(60)
pipeline.add_letter_grades()
stats = pipeline.get_statistics()

print(stats)
# Output: {
#     'total_students': 50,
#     'passed': 38,
#     'failed': 12,
#     'average_grade': 72.5,
#     'grade_distribution': {'A': 10, 'B': 15, 'C': 13, 'D': 7, 'F': 5}
# }

pipeline.export_results("processed_students.csv")
# Output: [TIMER] Processing took 0.15 seconds
#         [LOGGER] Processed 50 records
#         Results exported successfully
```