CPSC 250 – Programming for Data Manipulation – Test 2 Solution Key

Part A: Multiple Choice (10 points)

1. C. The original list is changed.
2. C. It initializes the object's state.
3. C. Frees memory by deleting objects with zero references.
4. C. Setter methods can include validation logic before updating a value.
5. D. `def __eq__(self, other):`

Part B: Find the Error (15 points)

1. The function `double` returns nothing. **Fix:** Add `return n` to return the result.
2. The constructor is not assigning values to `self.name` and `self.grade`. **Fix:** Use `self.name = name`, etc.
3. Default mutable argument is unsafe. **Fix:** Use `if names is None: names = []`.
4. Missing `self` in `__str__` method. **Fix:** `def __str__(self):`
5. The method returns a tuple, not a Vector. **Fix:** Return `Vector(self.x + v.x, self.y + v.y)`.

Part C: Code Writing (30 points)

1.

```
def modify_list(mylist):
    mylist.append(42)

nums = [1, 2, 3]
modify_list(nums)
print(nums)  # Output: [1, 2, 3, 42]
```

Explanation: Lists are mutable and passed by reference. Changes inside the function affect the original.

2.

```
class Book:
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages

    def get_title(self):
        return self.title

    def set_title(self, title):
        self.title = title

    def get_author(self):
        return self.author

    def set_author(self, author):
        self.author = author

    def get_pages(self):
        return self.pages

    def set_pages(self, pages):
        self.pages = pages

    def summary(self):
        return f"{self.title} by {self.author}, {self.pages} pages"

    def __str__(self):
        return f"Book: {self.title} by {self.author} ({self.pages} pages)"
```

3.

```
    def __add__(self, other):
        return Book("Collection", "Various", self.pages + other.pages)
```

Part D: Code Commentary (20 points)

```
class InventoryItem:                    # Defines a new class named InventoryItem
    def __init__(self, name, quantity):  # Constructor to initialize name and qua
        self.name = name                # Sets the item's name
        self.quantity = quantity        # Sets the initial quantity

    def restock(self, amount):          # Adds stock to the inventory
        self.quantity += amount         # Increases quantity by given amount

    def sell(self, amount):             # Sells items if enough in stock
        if amount > self.quantity:      # Checks for sufficient inventory
```

```
            print("Not enough in stock")  # Warns if not enough
        else:
            self.quantity -= amount       # Subtracts amount from inventory

    def __str__(self):                    # Returns a string representation
        return f"{self.name}: {self.quantity} in stock"  # Displays current stock
```