Python Advanced Concepts – Practice Set

This set is based on the topics we've covered so far:

- **Decorators in Python**
- **Getters and Setters**
- **Static & Class Methods**
- **Magic/Dunder Methods**
- **Exception Handling and Custom Errors**
- **map(), filter(), and reduce()**
- **Walrus Operator**
- **args and kwargs**

These exercises are designed to take your Python skills to the next level by practicing object-oriented and functional programming features.

---

1. Decorators in Python

1. Write a decorator `logger` that prints `"Function is being called"` before the function runs. Use it to decorate a function `say_hello()` that prints `"Hello!"`.
2. Write a decorator `timer` that calculates how long a function takes to execute. Test it with a function that sums numbers from 1 to 1,000,000.

---

2. Getters and Setters

1. Create a class `Employee` with a private attribute `_salary`.
    1. Use `@property` to define a getter for `salary`.
    2. Use `@salary.setter` to prevent setting negative values (print a warning instead).
    3. Create an object and test by setting positive and negative salaries.

---

3. Static & Class Methods

1. Create a class `MathUtils` with:
    1. A `@staticmethod` called `add(a, b)` that returns `a + b`.
    2. A `@classmethod` called `description(cls)` that prints `"This is a utility class for math operations."`
2. Call both methods without creating an object.

---

4. Magic/Dunder Methods

1. Create a class `Book` with attributes `title` and `author`.
    1. Implement `__str__()` so that printing the object displays `"Title by Author"`.
    2. Implement `__len__()` so that `len(book)` returns the length of the title.
2. Create two `Book` objects and test these methods.

---

5. Exception Handling and Custom Errors

1. Write a program that asks the user to enter a number and handles:
    1. `ValueError` if the input is not a number
    2. `ZeroDivisionError` if you try to divide by zero
2. Create a custom exception `NegativeNumberError` and raise it when the user enters a negative number.

---

6. map(), filter(), and reduce()

1. Use `map()` to convert `[1, 2, 3, 4, 5]` into their cubes.
2. Use `filter()` to get only even numbers from `[10, 11, 12, 13, 14]`.
3. Use `reduce()` from `functools` to find the product of all elements in `[1, 2, 3, 4]`.

---

7. Walrus Operator

1. Use the walrus operator to read input until the user enters `"quit"`. Print each input as it is entered.
2. Use the walrus operator in a list comprehension to store lengths of words from `["python", "rocks", "ai"]` in a list while filtering out words shorter than 4 characters.

---

8. *args and **kwargs

1. Write a function `sum_all(*args)` that accepts any number of integers and returns their sum.
2. Write a function `print_details(**kwargs)` that prints key-value pairs passed as arguments, for example:

```
print_details(name="Alice", age=25, city="Delhi")
# Output:
# name: Alice
# age: 25
# city: Delhi
```

---

9. Bonus Challenges

1. Combine a decorator with `*args` and `**kwargs` support so it can wrap any function regardless of its parameters.
2. Implement `__add__` in a `Vector` class so that adding two `Vector` objects returns a new `Vector` with summed components.
3. Create a small program where invalid user input raises a custom exception, logs the error, and continues execution instead of crashing.