# Python Problem Set Worksheet 1.G CMPE 261 Large Scale Programming

Osman Emre Tutay

December 2025

## Problem 1: Basic Inheritance

In large-scale machine learning frameworks, specific models often inherit from a generic base model class to gain common functionality. Write a Python script that models this relationship.

### Instructions:

- Define a parent class `BaseModel` with an `__init__` method that accepts `model_name` and `version`.
- Add a method `get_info()` to `BaseModel` that returns a string formatted as "Model: [name] (v[version])".
- Define a child class `NeuralNetwork` that inherits from `BaseModel`.
- Inside `NeuralNetwork`, add a new instance variable `layer_count` via its `__init__` method.
- Create an instance of `NeuralNetwork` with the name "ResNet", version `50`, and `152` layers.
- Print the result of `get_info()` (inherited method) and the `layer_count` (child attribute).

### Expected Output:

```
Loaded: Model: ResNet (v50)
Layers: 152
```

## Problem 2: Method Overloading

Write a Python script that calculates the area of a shape, behaving differently depending on whether one or two arguments are passed.

### Instructions:

- Create a class `AreaCalculator`.
- Define a method `calculate_area` that accepts `self`, `length`, and an optional argument `width` (set the default value of `width` to `None`).
- Inside the method, check if `width` is `None`.
  - If it is `None`, assume it is a square and return `length * length`.
  - If `width` is provided, assume it is a rectangle and return `length * width`.
- Instantiate the class and call the method twice: once with one argument (5) and once with two arguments (5, 10). Print the results.

### Expected Output:

```
Area of Square: 25
Area of Rectangle: 50
```

## Problem 3: Method Overriding

In a web infrastructure, different types of servers handle requests differently. A standard server might handle plain text, while a secure server must perform encryption checks first. Write a Python script to demonstrate this difference using method overriding.

### Instructions:

- Define a parent class `Server` with a method `handle_request()` that prints "Handling standard HTTP request...".
- Define a child class `SecureServer` that inherits from `Server`.
- Override the `handle_request()` method in `SecureServer`.
- Inside the overridden method, print two lines:
  1. "[Security] Verifying encryption certificates..."
  2. "Handling secure HTTPS request..."
- Create a list containing one instance of `Server` and one instance of `SecureServer`.
- Loop through the list and call `handle_request()` on each to see the polymorphic behavior.

### Expected Output:

```
Handling standard HTTP request...
[Security] Verifying encryption certificates...
Handling secure HTTPS request...
```

## Problem 4: Encapsulation

In computer graphics, a pixel's color is often represented by Red, Green, and Blue values ranging from 0 to 255. If a value goes outside this range, it causes visual artifacts. Write a Python class that uses encapsulation to protect these values from invalid assignments.

### Instructions:

- Create a class `Pixel`.
- In the `__init__` method, accept `r`, `g`, and `b` values. Assign them to private attributes: `__r`, `__g`, and `__b`.
- Create a public method `get_color()` that returns a tuple of the current values (e.g., `(255, 0, 0)`).
- Create a public method `set_red(new_r)` to modify the red channel.
- Inside `set_red`, add validation logic:
  - If `new_r` is between 0 and 255 (inclusive), update `__r`.
  - If `new_r` is invalid, print an error message like "Error: Invalid Red value." and do not change the internal state.
- Test your class:
  1. Create a `Pixel(100, 150, 200)`.
  2. Try to set the red value to 300 (invalid) and verify the error.
  3. Set the red value to 50 (valid).
  4. Print the final color using `get_color()`.

### Expected Output:

```
Error: Invalid Red value.
Final Pixel Color: (50, 150, 200)
```

## Problem 5: Interface

Write a Python script that defines a "Playable" interface for media devices.

## Instructions:

- Define a class `Playable` inheriting from `ABC`.
- Define two abstract methods: `play()` and `stop()`. Do not implement any logic inside them
- Create a class `MP3Player` that inherits from `Playable`. Implement both methods to print specific messages (e.g., "MP3 starting", "MP3 stopping").
- Create a class `VideoPlayer` that inherits from `Playable`. Implement both methods with different messages.
- Create a list containing an instance of each player. Loop through the list and call `play()` on each object.

## Expected Output:

```
Playing MP3 music...
Playing Video file...
```