**CODING OUTLINE**

**1. Class: AffirmationGenerator**

* **Purpose**: Manages affirmations, including categories and methods to retrieve affirmations.
* **Attributes**:
  + affirmations: A dictionary that stores affirmation categories as keys and lists of affirmations as values.
* **Methods**:
  + \_\_init\_\_(self): Initializes the dictionary with categories and their affirmations.
  + get\_random\_affirmation(self): Returns a random affirmation from any category.
  + get\_affirmation\_by\_category(self, category): Returns a random affirmation from a specific category.
  + list\_categories(self): Returns a list of all categories.

**2. Class: AffirmationApp**

* **Purpose**: Handles user interaction and overall program flow.
* **Attributes**:
  + generator: An instance of AffirmationGenerator.
* **Methods**:
  + \_\_init\_\_(self): Initializes the AffirmationGenerator instance.
  + display\_welcome\_message(self): Prints a welcome message to the user.
  + prompt\_user\_choice(self): Prompts the user to choose between a random affirmation or selecting a category.
  + display\_affirmation(self, choice): Displays an affirmation based on the user’s choice (random or by category).
  + choose\_category\_affirmation(self): Lists categories and handles user selection of a specific category.
  + run(self): Main method to run the program, orchestrating the flow by calling the appropriate methods based on user input.

**Program Flow**

1. **Initialize**:
   * Create an instance of AffirmationApp.
2. **Run the Application**:
   * Call the run() method in AffirmationApp to start the program.
3. **Display Welcome Message**:
   * display\_welcome\_message(): Shows a warm, motivational welcome message to the user.
4. **Prompt User for Choice**:
   * prompt\_user\_choice(): Asks the user if they want a random affirmation or a category-specific one.
   * Pass the user's choice to display\_affirmation().
5. **Display Affirmation**:
   * display\_affirmation(choice): If the user chooses a random affirmation, call get\_random\_affirmation() from AffirmationGenerator.
   * If the user wants to select a category, call choose\_category\_affirmation().
6. **Handle Category Selection**:
   * choose\_category\_affirmation(): Lists categories using list\_categories() from AffirmationGenerator.
   * Prompts the user to select a category by number, then displays an affirmation from that category.

**Benefits of This Outline**

* **Modularity**: Each class and method has a specific role, making the code easy to expand or modify.
* **User Interaction**: Clear, distinct methods for handling user input and displaying affirmations enhance the program’s readability and flow.
* **Error Handling**: The program includes basic error handling to guide users if they make an invalid choice.

**1. Classes**

* **AffirmationGenerator**: This class manages all affirmations, including storing categories, providing random affirmations, and affirmations by category.
* **AffirmationApp**: This class handles the user interface, prompts the user, and displays affirmations based on the user’s selection.

**2. Constructor (\_\_init\_\_ method)**

* Each class has an **\_\_init\_\_ method**, which is the **constructor** in Python. This method is called automatically when a new instance of the class is created, initializing the instance’s attributes.
* **AffirmationGenerator.\_\_init\_\_** initializes the affirmations dictionary, storing categories and affirmations.
* **AffirmationApp.\_\_init\_\_** initializes an instance of AffirmationGenerator and assigns it to the generator attribute.

**3. Instance Variables**

* **Instance variables** are variables specific to each instance of a class, set within the constructor (\_\_init\_\_) method.
* **In AffirmationGenerator**:
  + self.affirmations: This is an instance variable that holds the dictionary of categories and affirmations.
* **In AffirmationApp**:
  + self.generator: This is an instance variable that holds an instance of AffirmationGenerator for accessing affirmations.

**4. Class Variables**

* A **class variable** is shared among all instances of a class, rather than being unique to each instance. In our current code, we don’t have any class variables.
* **Example** (not in our code): If we added a variable like affirmation\_count that counts all affirmations across instances, it would be a class variable.

**5. Instances**

* An **instance** is an individual object created from a class.
* In AffirmationApp, self.generator is an instance of the AffirmationGenerator class.
* When app = AffirmationApp() is created in the if \_\_name\_\_ == "\_\_main\_\_": block, app is an instance of AffirmationApp.

**6. Methods**

* Methods are functions defined within a class that operate on instance variables.
* Both classes contain multiple methods to carry out specific tasks:
  + **AffirmationGenerator** has methods for managing affirmations (get\_random\_affirmation, get\_affirmation\_by\_category, list\_categories).
  + **AffirmationApp** has methods for handling user interaction and program flow (display\_welcome\_message, prompt\_user\_choice, display\_affirmation, choose\_category\_affirmation, run).

**Summary of Each Component in the Code**

* **Classes**: AffirmationGenerator, AffirmationApp
* **Constructor**: \_\_init\_\_ methods in both classes
* **Instance Variables**:
  + affirmations in AffirmationGenerator
  + generator in AffirmationApp
* **Class Variables**: None in the current code
* **Instances**:
  + self.generator as an instance of AffirmationGenerator
  + app as an instance of AffirmationApp
* **Methods**: Defined in each class to perform specific functions, such as retrieving affirmations or interacting with the user.