Activity 1: Design Your Own Class

Object-Oriented Programming (OOP) in Python, focusing on creating classes, using constructors, and implementing polymorphism.

Let's create a class called `Smartphone` that represents a smartphone with attributes and methods.

Code Example: (class Smartphone)

```
    def __init__(self, brand, model, storage, battery_life):
        """Initialize the smartphone with brand, model, storage, and battery life."""
        self.brand = brand
        self.model = model
        self.storage = storage  # in GB
        self.battery_life = battery_life  # in hours

    def make_call(self, contact):
        """Simulate making a call to a contact."""
        print(f"{self.brand} {self.model} is calling {contact}...")

    def take_photo(self):
        """Simulate taking a photo."""
        print(f"{self.brand} {self.model} is taking a photo!")

    def display_info(self):
        """Display smartphone information."""
        print(f"Brand: {self.brand}, Model: {self.model}, Storage: {self.storage}GB, Battery Life: {self.battery_life}
hours")

# Example of creating a Smartphone object
my_phone = Smartphone("Apple", "iPhone 14", 128, 20)
my_phone.display_info()
my_phone.make_call("Alice")
my_phone.take_photo()
```

Inheritance Example: Advanced Smartphone

Now, let's create a subclass that inherits from `Smartphone` to represent an advanced smartphone with additional features.

class Advanced Smartphone(Smartphone):

```
    def __init__(self, brand, model, storage, battery_life, camera_quality):
        """Initialize the advanced smartphone with camera quality."""
        super().__init__(brand, model, storage, battery_life)
        self.camera_quality = camera_quality  # in MP

    def take_photo(self):
        """Override the take_photo method to include camera quality."""
        print(f"{self.brand} {self.model} is taking a {self.camera_quality}MP photo!")

# Example of creating an Advanced Smartphone object
my_advanced_phone = AdvancedSmartphone("Samsung", "Galaxy S21", 256, 25, 108)
my_advanced_phone.display_info()
my_advanced_phone.take_photo()
```

Activity 2: Polymorphism Challenge

Now, let's create a program that includes different types of vehicles, each implementing a `move()` method differently.

Code Example:(class Vehicle)

```
    def move(self):
        """Generic move method for vehicles."""
        raise NotImplementedError("Subclasses must implement this method.")

class Car(Vehicle):
    def move(self):
        """Override move method for Car."""
        print("Driving 🚗")

class Plane(Vehicle):
    def move(self):
        """Override move method for Plane."""
        print("Flying ✈")

class Bicycle(Vehicle):
    def move(self):
        """Override move method for Bicycle."""
        print("Cycling 🚲")

# Function to demonstrate polymorphism
def demonstrate_movement(vehicles):
    for vehicle in vehicles:
        vehicle.move()

# Create instances of each vehicle
my_car = Car()
my_plane = Plane()
my_bicycle = Bicycle()
```

Demonstrate polymorphism

demonstrate_movement([my_car, my_plane, my_bicycle])

Explanation:
- Polymorphism is demonstrated through the **`move()`** method, which is defined in each subclass (`Car`, `Plane`, `Bicycle`) with different implementations.

- The **`demonstrate_movement`** function takes a list of vehicles and calls their **`move()`** method, showcasing how each vehicle behaves differently.

By following this structure, can effectively showcase the understanding of classes, inheritance, and polymorphism.