# Advanced Python Classes Worksheet

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Part 1: Recap and Warm-Up

Before starting, review the following key terms:

- Class: A template or blueprint for creating objects.  
- Object: An instance of a class (the real thing created from the class).  
- \_\_init\_\_: A special method used to initialise an object.  
- Attributes: Things the object has (e.g., name, colour, power).  
- Methods: Things the object can do (e.g., say\_hello).

Warm-Up:

1. Create a simple class for a car with attributes for make, model, and year.

2. Add a method to display the car’s details.

Example:  
car1 = Car("Toyota", "Corolla", 2020)  
car1.display\_details()

## Part 2: Inheritance

Inheritance allows one class to use the attributes and methods of another class.

Example:  
class Vehicle:  
 def \_\_init\_\_(self, make, model):  
 self.make = make  
 self.model = model  
   
 def display\_info(self):  
 print(f"Make: {self.make}, Model: {self.model}")  
  
class Car(Vehicle):  
 def \_\_init\_\_(self, make, model, year):  
 super().\_\_init\_\_(make, model)  
 self.year = year  
   
 def display\_car\_info(self):  
 print(f"Make: {self.make}, Model: {self.model}, Year: {self.year}")

Activity 1:

1. Create a parent class called `Animal` with attributes for name and species.

2. Create a child class called `Dog` that adds a `breed` attribute.

3. Add a method to make the dog bark. Example: 'Woof! I am a Labrador'.

## Part 3: Encapsulation

Encapsulation restricts access to certain parts of an object.

Example:  
class BankAccount:  
 def \_\_init\_\_(self, account\_holder, balance):  
 self.account\_holder = account\_holder  
 self.\_\_balance = balance # Private attribute  
  
 def deposit(self, amount):  
 self.\_\_balance += amount  
  
 def withdraw(self, amount):  
 if amount <= self.\_\_balance:  
 self.\_\_balance -= amount  
 print(f"Withdrew £{amount}. Remaining balance: £{self.\_\_balance}.")  
 else:  
 print("Insufficient funds.")  
  
 def display\_balance(self):  
 print(f"Balance: £{self.\_\_balance}")

Activity 2:

1. Create a class for a student with a private attribute for grades.

2. Add methods to add a grade and calculate the average grade.

3. Create a student object and add three grades. Display the average.

## Part 4: Polymorphism

Polymorphism allows methods to have the same name but behave differently based on the object.

Example:  
class Bird:  
 def make\_sound(self):  
 print("Tweet!")  
  
class Dog:  
 def make\_sound(self):  
 print("Woof!")  
  
# Polymorphism in action  
animals = [Bird(), Dog()]  
for animal in animals:  
 animal.make\_sound()

Activity 3:

1. Create two classes, `Cat` and `Cow`, both with a `make\_sound` method.

2. Use a loop to call `make\_sound` for a list of animals containing both `Cat` and `Cow`.

## Part 5: Advanced Challenge

Create a mini game with the following requirements:  
1. Create a parent class `Character` with attributes for name and health.  
2. Add methods to attack another character (reduce their health) and display the character’s stats.  
3. Create two child classes, `Hero` and `Villain`, with unique abilities (e.g., the hero can heal, the villain can poison).  
4. Write a script where the hero and villain fight until one of them loses all health.

## Reflection Questions

1. What new concepts did you learn from this worksheet?

2. How would you explain inheritance or polymorphism to someone else?

3. Can you think of real-world examples that could use these concepts?