Encapsulation and Abstraction in Python

# 1. Encapsulation

Definition:  
Encapsulation is the bundling of data (attributes) and methods (functions) that operate on that data into a single unit (class), while also restricting direct access to some of the object’s components (using private or protected access).

Example:

class BankAccount:  
 def \_\_init\_\_(self, balance):  
 self.\_\_balance = balance # private variable  
  
 def deposit(self, amount):  
 self.\_\_balance += amount  
  
 def get\_balance(self):  
 return self.\_\_balance  
  
account = BankAccount(1000)  
account.deposit(500)  
print(account.get\_balance())

Line-by-line Explanation:

* class BankAccount: -> Defines a class to represent a bank account.
* def \_\_init\_\_(self, balance): -> Constructor method to initialize balance.
* self.\_\_balance = balance -> Private variable using double underscore.
* def deposit(self, amount): -> Method to add money to the account.
* self.\_\_balance += amount -> Adds amount securely to balance.
* def get\_balance(self): -> Method to retrieve current balance.
* account = BankAccount(1000) -> Creates an object with balance 1000.
* account.deposit(500) -> Adds 500 to the balance.
* print(account.get\_balance()) -> Prints updated balance (1500).

# 2. Abstraction

Definition:  
Abstraction means hiding complex implementation details and only exposing essential features. In Python, this is achieved using abstract base classes (ABC module).

Example:

from abc import ABC, abstractmethod  
  
class Vehicle(ABC):  
 @abstractmethod  
 def start\_engine(self):  
 pass  
  
class Car(Vehicle):  
 def start\_engine(self):  
 print("Car engine started.")  
  
car = Car()  
car.start\_engine()

Line-by-line Explanation:

* from abc import ABC, abstractmethod -> Import abstract base class tools.
* class Vehicle(ABC): -> Abstract class using ABC module.
* @abstractmethod -> Marks start\_engine as a method that must be implemented.
* def start\_engine(self): -> Abstract method with no implementation.
* class Car(Vehicle): -> Concrete class that inherits from Vehicle.
* def start\_engine(self): -> Implements the abstract method.
* car = Car() -> Creates an object of Car.
* car.start\_engine() -> Calls the implemented method.