**Notes on OOP Concepts in Python (Banking System Example)**

**1. Encapsulation**

**Definition:**

* **Encapsulation** is the process of **hiding** data (attributes) inside a class and restricting direct access from outside.
* It is achieved using **private attributes (\_\_attribute)** and **getter methods**.

**Key Points:**

**Protects sensitive data** (e.g., balance).  
 **Allows controlled access** using methods.  
 **Prevents accidental modification** of critical attributes.

**Example in Banking:**

* The **balance** of an account is private, meaning no one can change it directly.
* Access is allowed only through **getter (get\_balance()) and setter methods**.

**2. Inheritance**

**Definition:**

* **Inheritance** allows a child class to **reuse** attributes and methods from a parent class.
* It helps in **code reusability and reducing duplication**.

**Key Points:**

**Child classes inherit properties** of the parent class.  
**Promotes hierarchical relationships** (e.g., a **SavingsAccount** is a specialized **BankAccount**).  
**Reduces redundancy** by reusing existing code.

**Example in Banking:**

* **SavingsAccount and CurrentAccount inherit from BankAccount**.
* They share common functionality (**deposit, withdraw, balance**) but also have unique features (**interest, overdraft**).

**3. Polymorphism**

**Definition:**

* **Polymorphism** allows methods with the **same name** to behave **differently** in different classes.
* It is useful for **overriding** methods in child classes.

**Key Points:**

**Same method name but different behavior**.  
**Helps in creating flexible and extensible programs**.  
**Makes it easier to add new features without modifying existing code**.

**Example in Banking:**

* The show\_details() method is **overridden** in **SavingsAccount and CurrentAccount** to display different account information.
* The withdraw() method behaves differently for **CurrentAccount** (allows overdraft) compared to **SavingsAccount** (no overdraft).

**4. Abstraction**

**Definition:**

* **Abstraction** is the concept of **hiding complex implementation details** and exposing only the necessary parts.
* Implemented using **Abstract Base Classes (ABC)** in Python.
* It **forces child classes to implement certain methods**.

**Key Points:**

**Hides unnecessary implementation details**.  
**Forces subclasses to follow a structure**.  
**Simplifies program complexity**.

**Example in Banking:**

* The **calculate\_interest() method is abstract** in BankAccount, meaning every account type **must implement its own version**.
* **SavingsAccount provides an interest rate**, whereas **CurrentAccount does not have interest**—both implement calculate\_interest() differently.

**5. Arguments & Keyword Arguments**

**Definition:**

* Arguments are **values passed to a function or constructor** when an object is created.
* **Positional arguments** must be passed in order.
* **Keyword arguments** allow specifying parameter names for clarity.
* **Default arguments** provide a default value if no value is given.

**Key Points:**

**Positional arguments** → Must be passed in the correct order.  
**Keyword arguments** → Improve readability.  
**Default arguments** → Provide flexibility.

**Example in Banking:**

* SavingsAccount("Alice", balance=1000, interest\_rate=4)
  + Alice is a **positional argument**.
  + balance=1000 and interest\_rate=4 are **keyword arguments** for clarity.
* balance=0 is a **default argument**, meaning if no balance is given, it starts at 0.