📘Python Encapsulation – Detailed Notes with Real-Time Examples

🕰️1. Introduction

- **Encapsulation** is the concept of **hiding internal data** of a class from outside access.
- Protects object integrity by controlling access to variables and methods.
- Achieved using **private (__var)** and **protected (_var)** variables.

🕰️2. Access Modifiers in Python

1. **Public**: Accessible from anywhere.
2. **Protected (_var)**: Accessible within class and subclasses.
3. **Private (__var)**: Accessible only within class.

🕰️3. Example: Public, Protected, Private Variables

```python
class Employee:
    def __init__(self, name, salary):
        self.name = name          # Public
        self._department = "IT"  # Protected
        self.__salary = salary    # Private

emp = Employee("Alice", 5000)

# Accessing public variable
print(emp.name)  # Alice

# Accessing protected variable
print(emp._department)  # IT (not recommended, but possible)

# Accessing private variable using name mangling
print(emp._Employee__salary)  # 5000
```

Real-Time: Protect sensitive employee data like salary or social security number from direct modification.

🕰️4. Using Getter and Setter Methods

- Encapsulation is best implemented with **getter** and **setter** methods.

```python
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.__salary = salary

    # Getter
    def get_salary(self):
        return self.__salary

    # Setter
    def set_salary(self, amount):
        if amount > 0:
            self.__salary = amount
        else:
            print("Invalid salary")

emp = Employee("Bob", 4000)
print(emp.get_salary())  # 4000
emp.set_salary(4500)
print(emp.get_salary())  # 4500
emp.set_salary(-100)     # Invalid salary
```

Real-Time: Ensures salary cannot be set to invalid negative values in **HR management systems**.

🕰️5. Example: Bank Account

```python
class BankAccount:
    def __init__(self, acc_no, balance):
        self.acc_no = acc_no
        self.__balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount

    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Insufficient balance")

    def get_balance(self):
        return self.__balance
```

```
acc = BankAccount(12345, 1000)
acc.deposit(500)
print(acc.get_balance())  # 1500
acc.withdraw(2000)        # Insufficient balance
```

Real-Time: Prevents direct access to account balance in **banking software**, ensuring secure transactions.

🕰️6. Example: Online Shopping Cart

```
class ShoppingCart:
    def __init__(self):
        self.__items = []  # Private list of items

    def add_item(self, item):
        self.__items.append(item)

    def remove_item(self, item):
        if item in self.__items:
            self.__items.remove(item)

    def view_cart(self):
        return self.__items

cart = ShoppingCart()
cart.add_item("Laptop")
cart.add_item("Mouse")
print(cart.view_cart())  # ['Laptop', 'Mouse']
cart.remove_item("Mouse")
print(cart.view_cart())  # ['Laptop']
```

Real-Time: Protects cart items from direct modification in **e-commerce applications**.

🕰️7. Summary

- Encapsulation **hides internal state** of objects.
- Use **private variables** and **getter/setter** methods.
- Protects sensitive data and enforces validation rules.
- **Real-Time Usage:** Employee salary management, Bank accounts, Shopping cart systems.

📥Download Options

- Download as DOCX
- Download as TXT
- Download as PDF