Advanced Python

Subject: Polymorphism and Encapsulation in Python OOP

Lecturer : Reza Akbari Movahed

Polymorphism and Encapsulation in Python OOP

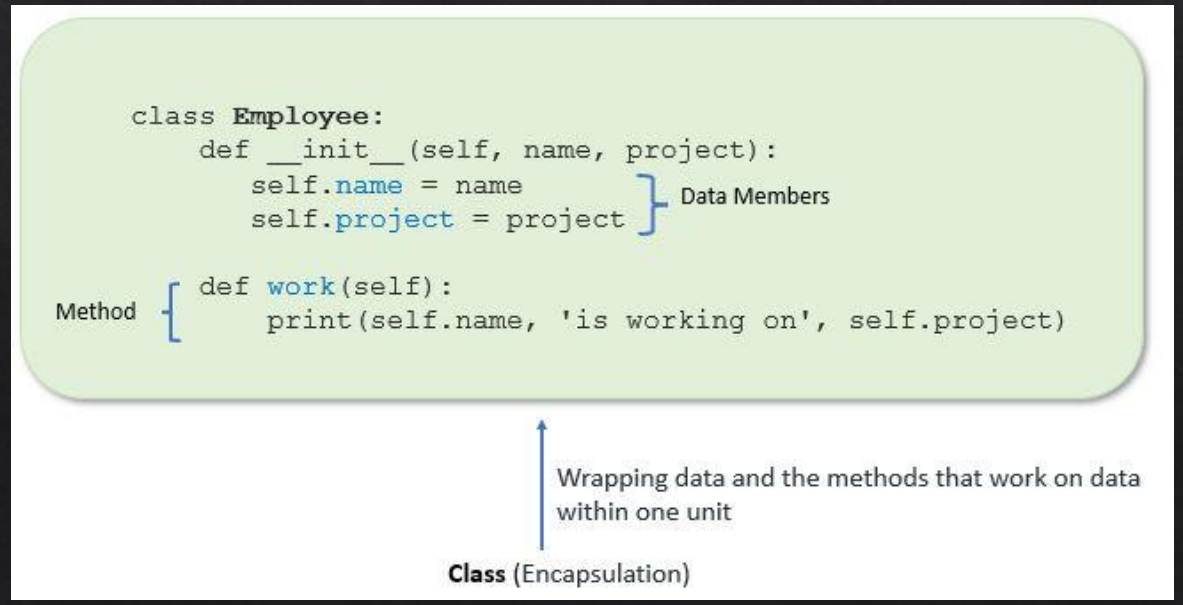
Polymorphism

- The word polymorphism means having many forms.
- In programming, polymorphism means the same function name (but different signatures) being used for different types.

Polymorphism and Encapsulation in Python OOP

Encapsulation concept

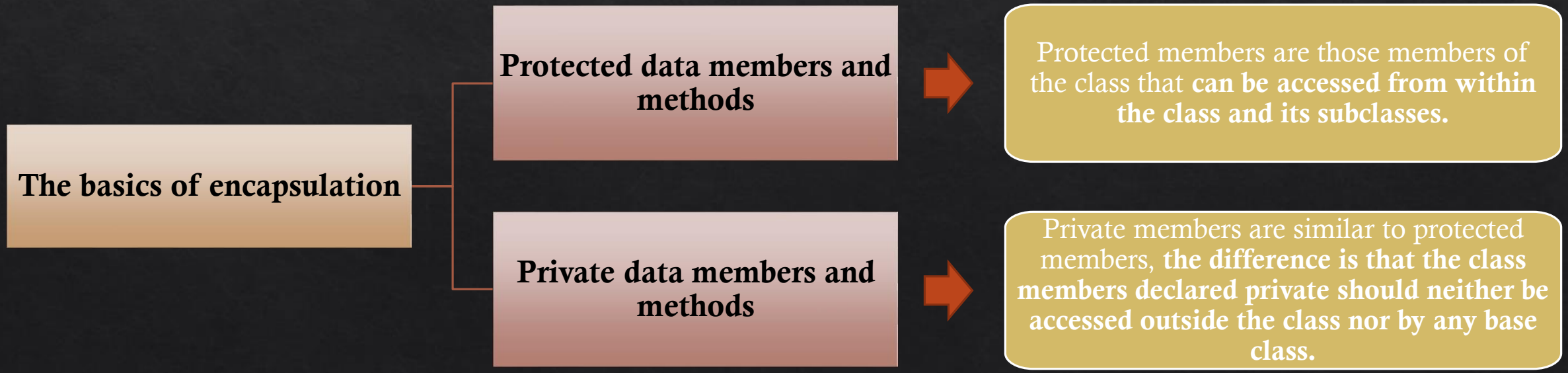
- Encapsulation in Python describes the concept of bundling data and methods within a single unit.
- So, for example, when you create a class, it means you are implementing encapsulation.
- A class is an example of encapsulation as it binds all the data members (instance variables) and methods into a single unit.



Polymorphism and Encapsulation in Python OOP

Encapsulation details

- Using encapsulation, we can hide an object's internal representation from the outside (information hiding).
- Also, encapsulation allows us to restrict accessing variables and methods directly and prevent accidental data modification by creating **private or protected data members and methods** within a class.
- Encapsulation is a way to can restrict access to methods and variables from outside of class.



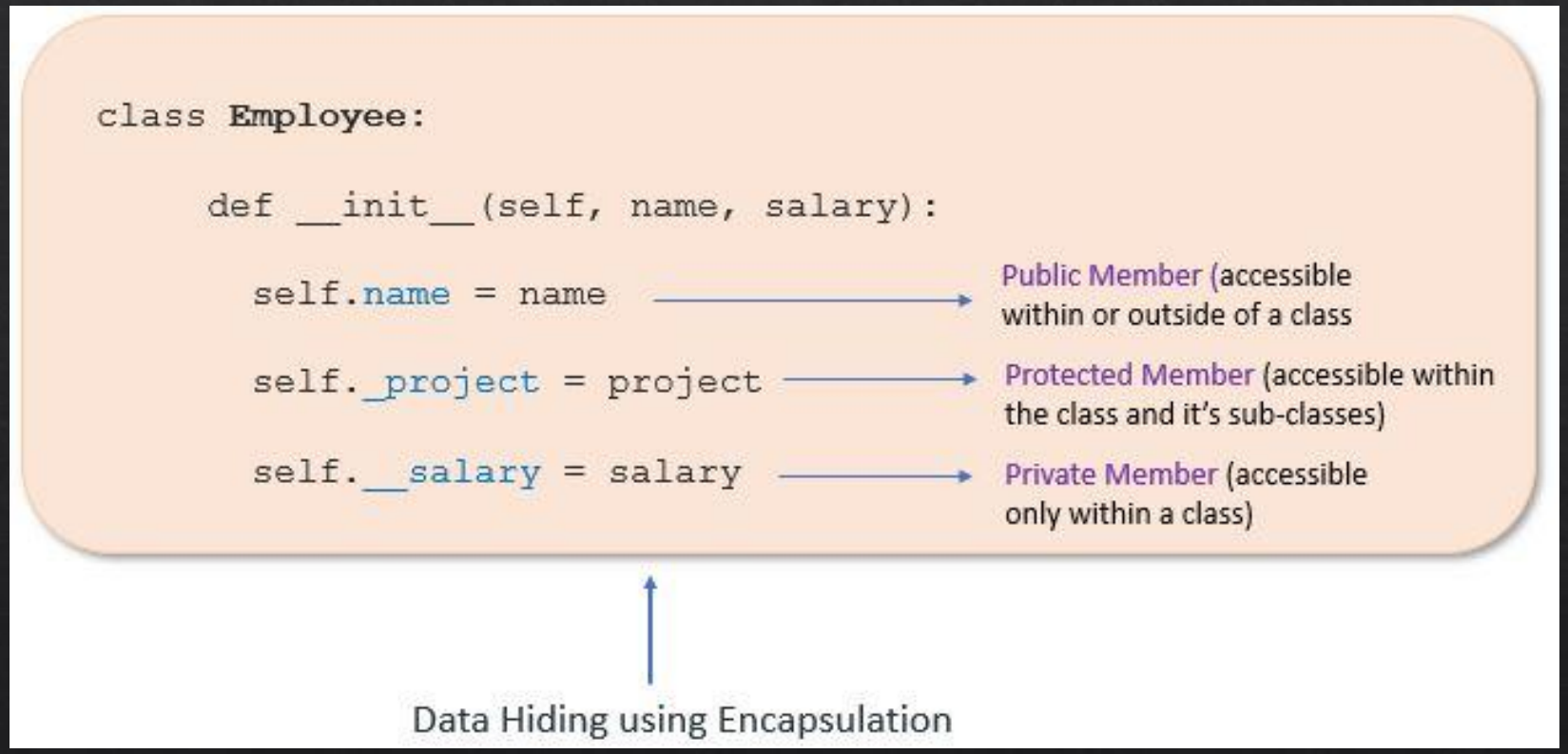
Polymorphism and Encapsulation in Python OOP

Access Modifiers in Python

- Encapsulation can be achieved by declaring the data members and methods of a class either as private or protected.
- Access modifiers limit access to the variables and methods of a class. Python provides three types of access modifiers private, public, and protected.
  - **Public Member**: Accessible anywhere from outside of the class.
  - **Private Member**: Accessible within the class
  - **Protected Member**: Accessible within the class and its sub-classes

Polymorphism and Encapsulation in Python OOP

Access Modifiers in Python



Polymorphism and Encapsulation in Python OOP

Private Member

- We can protect variables in the class by marking them private.
- To define a private variable add two underscores (__) as a prefix at the start of a variable name.
- Private members are accessible only within the class, and we can't access them directly from the class objects.
- We can access private members from outside of a class using the following two approaches
  - Create public method to access private members
  - Use name mangling

Polymorphism and Encapsulation in Python OOP

Protected Member

- Protected members are accessible within the class and also available to its sub-classes.
- To define a protected member, prefix the member name with a single underscore (_).
- Protected data members are used when you implement inheritance and want to allow data members access to only child classes.