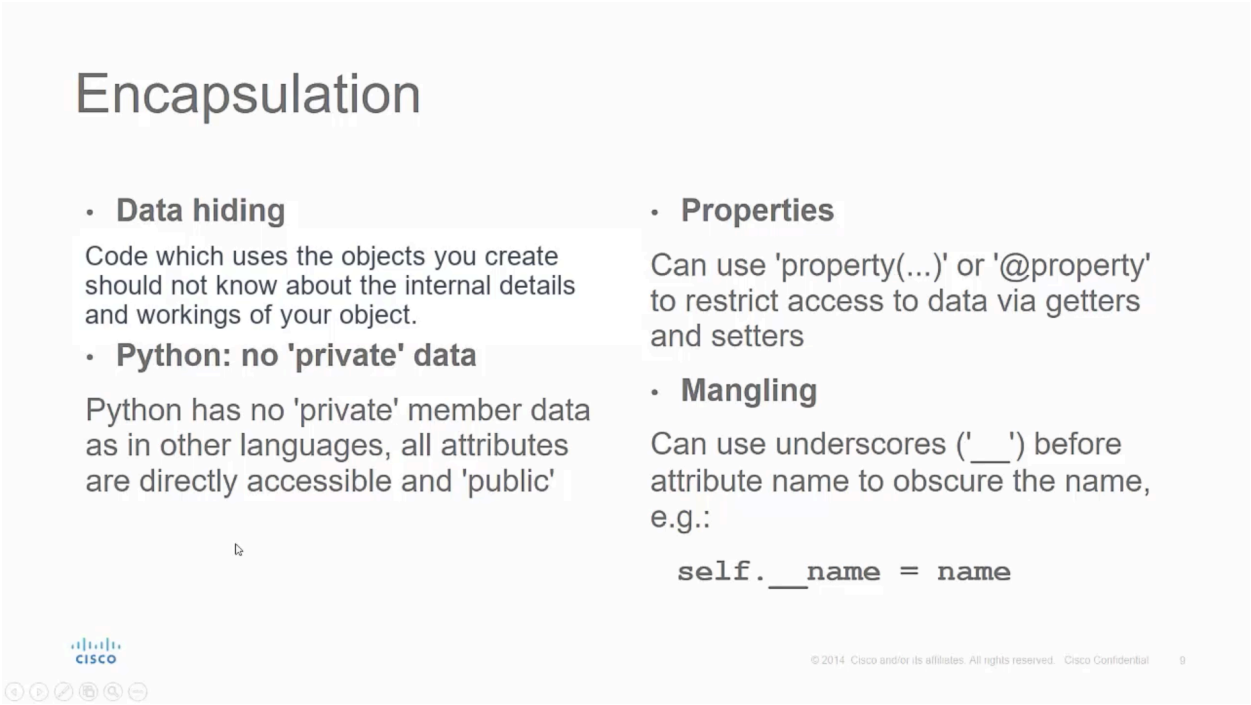


One of the concepts of OOP is the principle of encapsulation, also known as data hiding. The general idea of the principle is that code which uses the objects you create should not need to know about the internal details and workings of your object – the code should be able to access your methods to perform whatever functionality that method offers.

Some languages take the idea of encapsulation a step further, and make it possible to absolutely prohibit other code from accessing or changing internal items within an object. These private attributes and private methods cannot be accessed by any code other than the code of the object itself.

In Python, there are no 'private' attributes or members. Every attribute of the object you have created is visible and accessible to the outside world. If this is a problem, Python provides workarounds to help achieve the hiding of your data:

- **Properties:** Python provides the concept of making your data into a 'property', which makes it inaccessible to outside code. There are two methods of doing this:
  - `property(..) function`: Using the property function, it is possible to declare certain attributes as only accessible via getter and setter methods, which allow outside code to access the attribute via methods that are implemented in the object – insuring that the attributes are protected from invalid values.
  - `@property decorator`: Using the property decorator, it is possible to declare attributes as only accessible via a getter function by the same name. The decorator also allows for setting of the attribute.
- **Mangling:** It is also possible to 'mangle' the name of your attributes, by prefacing the attribute name with double underscores. Mangling tells Python that the attribute is intended to be private, and not directly accessible by outside code.

Neither of these methods entirely secures your classes from malicious outside access, but they do help prevent some inadvertent problems that might arise.