Object Oriented Programming using Python (I)

Lecture(11)

Encapsulation and Abstraction

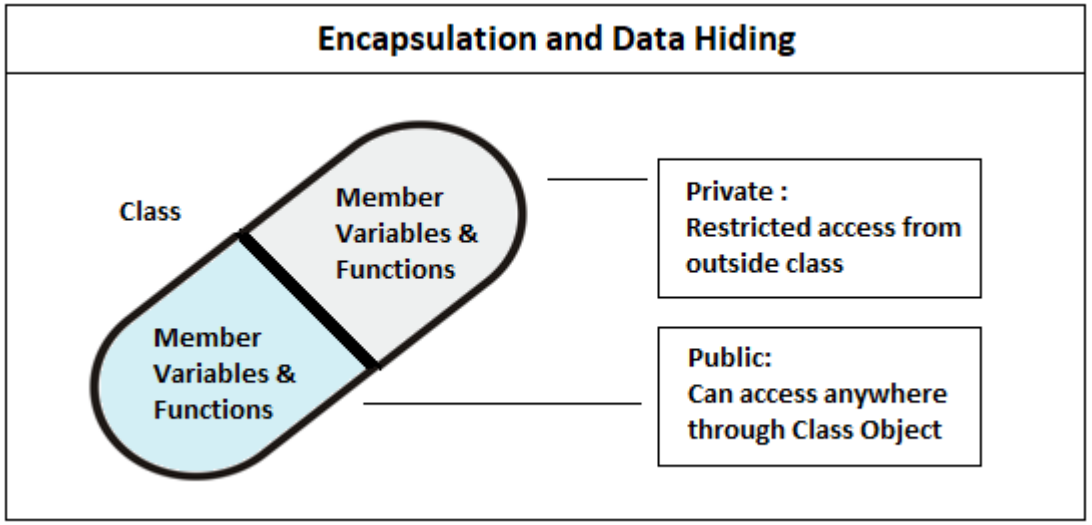
Prepared by Ahmed Eskander Mezher

University of Information Technology and Communications

College of Business Informatics

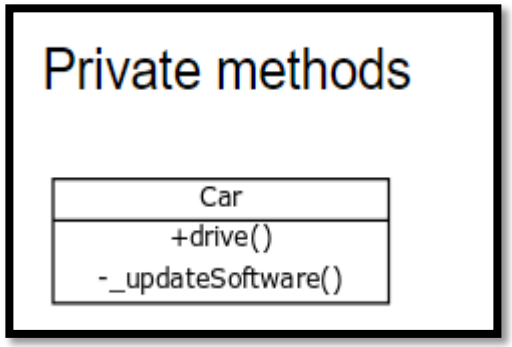
Encapsulation

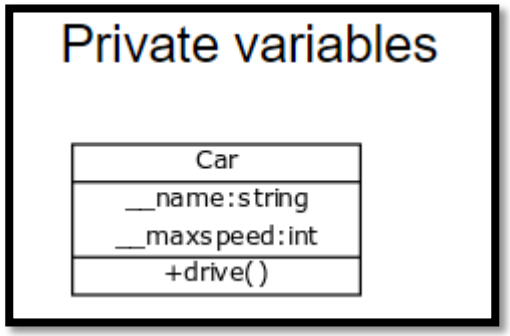
- In an object oriented python program, you can *restrict access* to methods and variables. This can prevent the data from being modified by accident and is known as *encapsulation*.
- The packaging of data and functions into single unit called encapsulation
- Data is not accessible to outside world and only functions wrapped in class can access it



Practical Encapsulation example

- Encapsulation Restricted access to methods or variables as shown in UML :



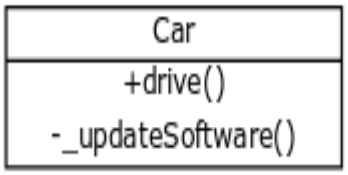


We create a class Car which has two methods: **drive()** and **updateSoftware().** When a car object is created, it will call the private methods __updateSoftware().

program

```
class Car:

    def __init__(self):
        self.__updateSoftware()

    def drive(self):
        print('driving')

    def __updateSoftware(self):
        print('updating software')

redcar = Car()
redcar.drive()
redcar.__updateSoftware() # not accesible from object.
```

Private methods



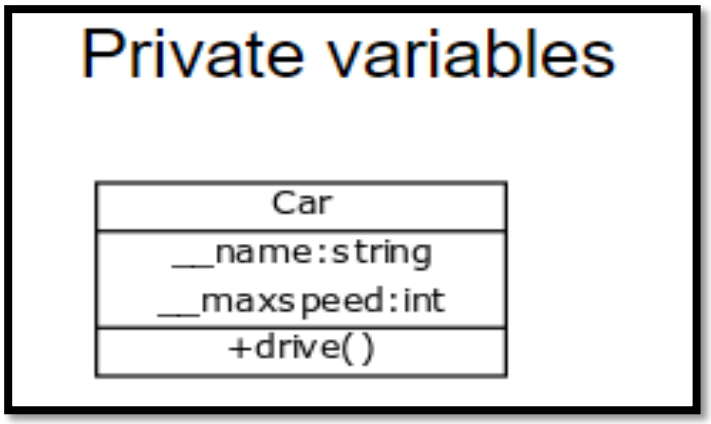
output

```
================== RESTART: C:/Python37/encapsulation.py ======
updating software
driving
Traceback (most recent call last):
  File "C:/Python37/encapsulation.py", line 14, in <module>
    redcar.__updateSoftware() # not accesible from object.
AttributeError: 'Car' object has no attribute '__updateSoftware'
>>>
```

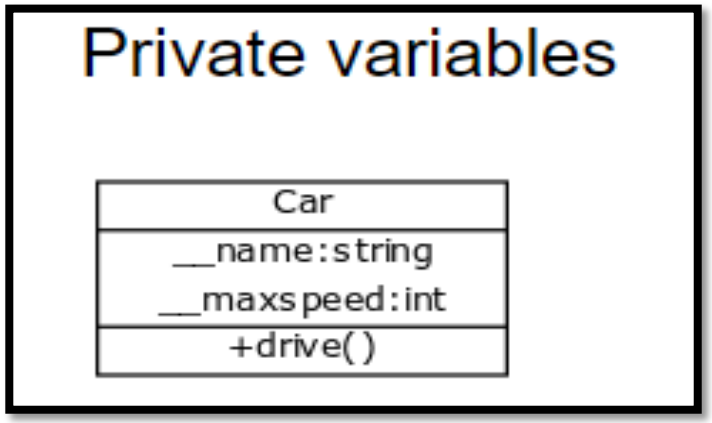
Class with private variables

Variables can be private which can be useful on many occasions. A private variable can only be changed within a class method and not outside of the class.

Objects can hold crucial data for your application and you do not want that data to be changeable from anywhere in the code .



```
class Car:

    __maxspeed = 0
    __name = ""

    def __init__(self):
        self.__maxspeed = 200
        self.__name = "Supercar"

    def drive(self):
        print('driving. maxspeed ' + str(self.__maxspeed))

redcar = Car()
redcar.drive()
redcar.__maxspeed = 10  # will not change variable because its private
redcar.drive()
```

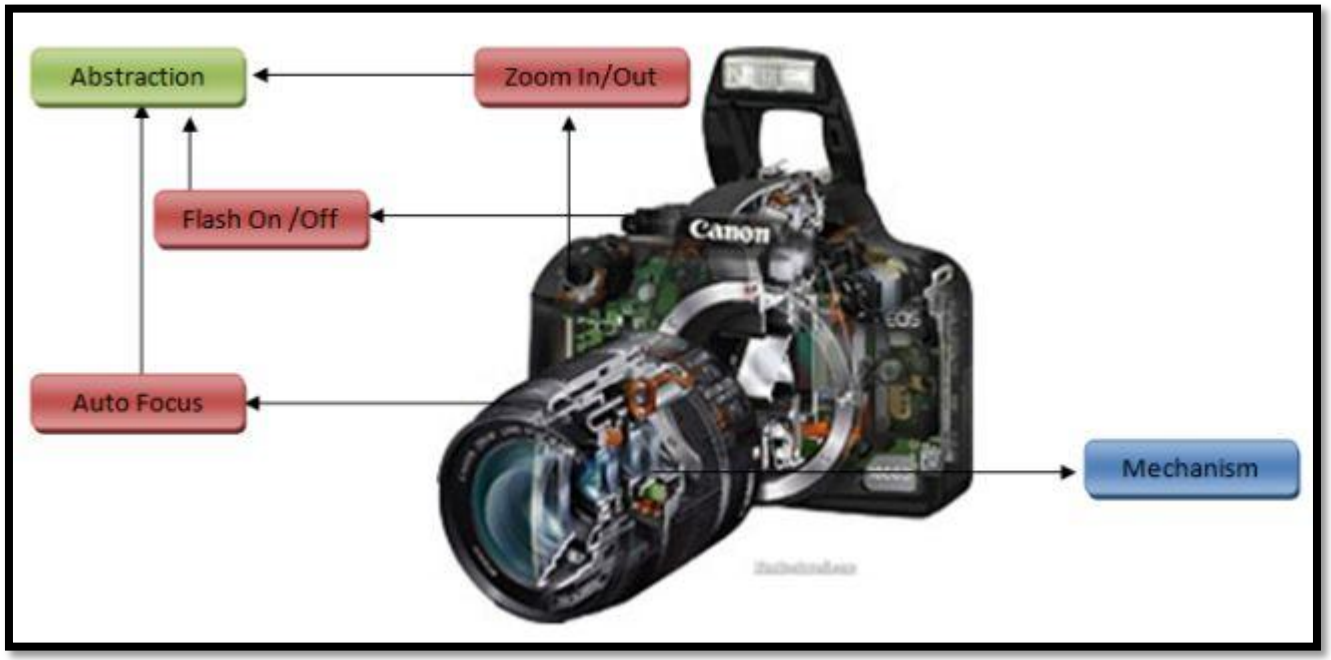


Note

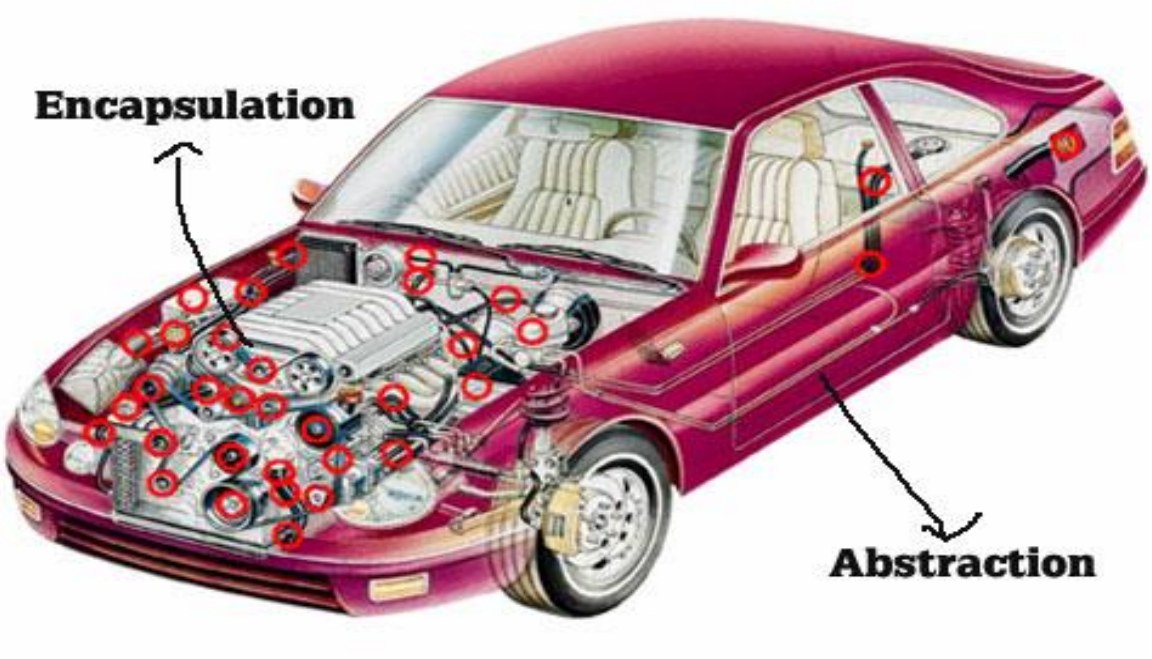
Encapsulation gives you more control over the degree of coupling in your code, it allows a class to change its implementation without affecting other parts of the code.

Abstraction

- Data abstraction refers to, providing only essential information hiding their background details
- i.e., to represent the needed information in program without presenting the details
- In simple words, abstraction is 'Hiding implementation details behind the interface' When abstraction is created our concern is limited to what it can do rather how it is doing it
- Eg:database system hides certain details how data is stored and created and maintained


Abstraction
Zoom In/Out
Flash On /Off
Auto Focus
Mechanism
Canon
EOS


Encapsulation
Abstraction


Encapsulation
Abstraction

Abstraction	Encapsulation
1. Abstraction solves the problem in the design level.	1. Encapsulation solves the problem in the implementation level.
2. Abstraction is used for hiding the unwanted data and giving relevant data.	2. Encapsulation means hiding the code and data into a single unit to protect the data from outside world.
3. Abstraction lets you focus on what the object does instead of how it does it	3. Encapsulation means hiding the internal details or mechanics of how an object does something.
4. **Abstraction**- Outer layout, used in terms of design. For Example:- Outer Look of a Mobile Phone, like it has a display screen and keypad buttons to dial a number.	4. **Encapsulation**- Inner layout, used in terms of implementation. For Example:- Inner Implementation detail of a Mobile Phone, how keypad button and Display Screen are connect with each other using circuits.