Getter & Setter methods

Access modifiers

Access modifiers is a general concept in Object Oriented Programming.

Access modifiers are classified in 3 categories in general - `private`, `public`, `protected`.

Access modifiers are used to restrict access to the variables and methods of the class. Most programming languages has three forms of access modifiers - `private`, `public`, `protected`.

Python doesn't have modifiers like -

- `private`
- `protected`
- `public`

We emulate the behaviour of access modifiers in Python either by using single underscore OR double underscore.

NOTE -

- **Using single underscore `_` is a convention**
- **Using double underscore `__` is NOT a convention.** (it leads to "name-mangling" by interpreter - adding class name in front of attribute)

```
class Employee:
    company = "Accenture"
```

```
    def __init__(self, name, designation, salary):
        self.name = name    # public
        self._designation = designation   # protected

        # confidential
        self.__salary = salary  # private


    # Getter METHOD
    def get_designation(self):
        return self._designation


    # Setter METHOD
    def set_designation(self, new_designation):
        self._designation = new_designation


# IF you execute following code line-by-line in following manner.

>>> emp1 = Employee("Virat Kohli", "Sr. Manager", 1000000)
>>> print(emp1.name)
>>> print(emp1._designation)

>>> print(emp1.__salary)  # <-- THIS LINE WILL PRODUCE FOLLOWING ERROR

AttributeError: 'Employee' object has no attribute '__salary'
```

- Single underscore `_` attributes should have a getter and setter method in order to access / modify them.
- This is not a compulsion but a convention in Python.

Can you do `dir(emp1)`?

You will still see the the attribute with double underscore (let's call it `private` attribute).

But the name of `private` attribute is mangled with class name `_<class-name>__<attr_name>`

NOTE

This is intentionally done in Python because it's NOT the ideal way to access such attribute.
They should be accessed using `@property` method.

```
>>> dir(emp1)

[
'_Employee__salary',  # <-- THIS IS NAME MANGLING. WE CAN STILL ACCESS WITH DOT
....,
....,

'_designation',
 'company',
'get_designation',
'name',
'set_designation']
```

QUOTE from `ZEN OF PYTHON`

> ***"There should be one-- and preferably only one --obvious way to do it." - Zen of Python***

- Double underscore `__` attributes should have a `@property` getter and `@property` setter method in order to access / modify them.
- This is kind of a compulsion (although we can use mangled attribute name and access attribute).

Example -

```
class Employee:
    company = "accenture"

    def __init__(self, name, designation, salary):
        self.name = name
        self._designation = designation
        self.__salary = salary

    # GETTER  ( CONVENTION )
    def get_designation(self):
        return self._designation

    # SETTER ( CONVENTION )
    def set_designation(self, new_designation):
        self._designation = new_designation

    @property
    def salary(self):  # GETTER
        return self.__salary

    @salary.setter  # SETTER
    def salary(self, new_value):
        self.__salary = new_value

    @property
    def compensation(self):  # GETTER
        return str(self.__salary / 78) + " USD"

    @compensation.setter
    def compensation(self, new_value):
        self.__salary = new_value


emp1 = Employee("Hardik", "Executive Director", 9000000)

emp1.compensation = 80  # SET
print(emp1.compensation) # GET
print(emp1.salary)
```

Conclusion

So properties are not just a replacement for getters and setters!

For the users of a class, properties are syntactically identical to ordinary attributes.