Python - property() function

The `property()` function is used to define properties in the Python class.

Consider the following Python script which defines the `person` class as having the getter and setter methods. The `getname()` method returns the value of the private instance attribute `__name`, while the `setname()` method assigns the value to the `__name` attribute.

Example: person.py

```
class person:
    def __init__(self, name="Guest"):
        self.__name=name
    def setname(self, name):
        self.__name=name
    def getname(self):
        return self.__name
```

The following interactive session shows the implementation of the person class from the above script.

```
>>> from person import person
>>> p1=person()
>>> p1.getname()
'Guest'
>>> p1.setname('Bill')
>>> p1.getname()
'Bill'
```

As you can see above, the `p1.getname()` method returns the value of attribute `__name` and the `setname()` method assigns a value to it. However, it would be nice if the getter and setter functions were called implicitly when we access an attribute, just like a property in Java and C#. This is where Python's built-in `property()` function comes in handy.

The `property()` method in Python provides an interface to instance attributes. It encapsulates instance attributes and provides a property, same as Java and C#.

The `property()` method takes the get, set and delete methods as arguments and returns an object of the `property` class.

Signature:
prop=property(getter, setter, deleter, docstring)

The following example demonstrates how to create a property in Python using the `property()` function.

Example: property() function

```
class person:
```

```
    def __init__(self):
        self.__name=''
    def setname(self, name):
        print('setname() called')
        self.__name=name
    def getname(self):
        print('getname() called')
        return self.__name
    name=property(getname, setname)
```

In the above example, `property(getname, setname)` returns the property object and assigns it to `name`. Thus, the `name` property hides the private instance attribute `__name`. The `name` property is accessed directly, but internally it will invoke the `getname()` or `setname()` method, as shown below.

```
>>> from person import person
>>> p1=person()
>>> p1.name="Steve"
setname() called
>>> p1.name
getname() called
'Steve'
```

As you can see above, the `getname()` method gets called automatically when we access the `name` property. In the same way, the `setname` method gets called when we assign a value to the `name` property. It also hides the instance attribute `__name`.

In the same way, you can specify a deleter method for the property, as shown in the below script.

Example: property() function

```
class person:
    def __init__(self, name):
        self.__name=name
    def setname(self, name):
        print('setname() called')
        self.__name=name
    def getname(self):
        print('getname() called')
        return self.__name
    def delname(self):
        print('delname() called')
        del self.__name
    name=property(getname, setname, delname)
```

The `delname()` function would be invoked when you delete the `name` property.

```
>>> from person import person
>>> p1=person()
>>> p1.name="Steve"
setname() called
```

```
>>> del p1.name
delname() called
```

In this way, we can define a property in the class using the `property()` function in Python.

@property decorator makes it easy to declare a property instead of calling the `property()` function.