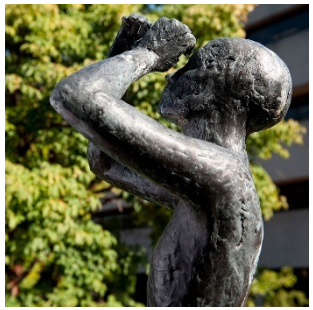

Lecture 5 – Writing a Class

CS2513

Cathal Hoare

A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Lecture Contents

Defining Properties

Getters and Setters - Version 1

- Traditionally in OO we set instance variables with ‘setters’ and read instance variables with ‘getters’
- These act on just one instance variable at a time, so we need to provide a getter/setter for each of _name, _job and _pay
- Beyond this, these are regular methods

Getters and Setters - Version 1

- Lets see a getter and setter for the name attribute:

```
class Person:
        def __init__(self, name, job, pay):
         ...
         def getName(self):
                        return self._name

        def setName(self, name):
                        if len(name) <= 0 or not name.isalpha():
                                        print("%s is not legal” % (name))
                        else:
                                        self._name = name

if __name__ == ‘__main__’:
        cathal = Person('Cathal', 'dev', 70000)
        cathal.setName(‘CathalNew’)
        print(cathal.getName())
```

- *See person_gettersetter.py for full code*

Getters and Setters - Version 1

- Assume that we add a getter and setter for the other instance variables in exactly the same way.
- Some would argue that the addition of methods in this way complicates third party code
- OOP insists that these methods are required.
- Python offers more freedom. PEP8 suggests that for simple attributes direct access is under certain circumstances permissible.
- But if we use direct access, third party developers' code is tightly bound to ours. What if we need to add error checking or we change some other aspect of our implication and this requires their code to change?

Getters and Setters - Version 2

- We modify our code to use properties - this allows access to our attributes through methods but using a syntax that is the same as when we directly access instance variables
- We can write print(cathal.name) but this will use the getName() method to access the instance variable.
- We can write cathal.pay = 100 but this will use the setPay(newpay) method to set the instance variable.

Getters and Setters - Version 2

- All of the simplicity of the attribute syntax but with all of the advantages of using the getter/setter methods:
  - We can have error checking or other processing as part of the getter/setter.
  - Third party code remains loosely bound to our code.
- In the event that direct access was originally implemented, we can now modify our code to use getters/setters without requiring changes in third party code.
- We can use two techniques - properties or property decorators.

Property Decorators Example

```
class Person:

    def __init__(self, name, job, pay):
        self._name = name
        self._job = job
        self._pay = pay

    @property
    def pay(self):    #Use name of attribute but without '_'

        return self._pay

    @pay.setter
    def pay(self, pay):    #Use name of attribute but without '_'

        if pay < 0 or pay > 100000:
            print("%i is an invalid pay value - no value set" % (pay))
            #TODO: Add Error Handling with Exceptions
        else:
            self._pay = pay


if __name__ == '__main__':
    cathal = Person('Cathal', 'dev', 70000)
    cathal.pay = 76000
    print(cathal.pay)
```

See person_decorator.py for full code

Properties Example

See person_property.py for full code

```
class Person:

    def __init__(self, name, job, pay):
        …

    def getPay(self):

        return self._pay

    def setPay(self, pay):

        if pay < 0 or pay > 100000:
            print("%i is an invalid pay value - no value set" % (pay))
            #TODO: Add Error Handling with Exceptions
        else:
            self._pay = pay

    pay = property(getPay, setPay) #Note order of methods in list - can also have
                                   #del and doc


if __name__ == '__main__':
    cathal = Person('Cathal', 'dev', 70000)
    cathal.pay = 76000
    print(cathal.pay)
```

Classes

- Create a class that represents a light.
- A light is described by its power (watts) and whether it is on or off.
- A light can change between on and off states.
- A light's wattage can be modified.
- Provide a means of printing the state of a light.


Next Time:
Developing Our Classes Futher