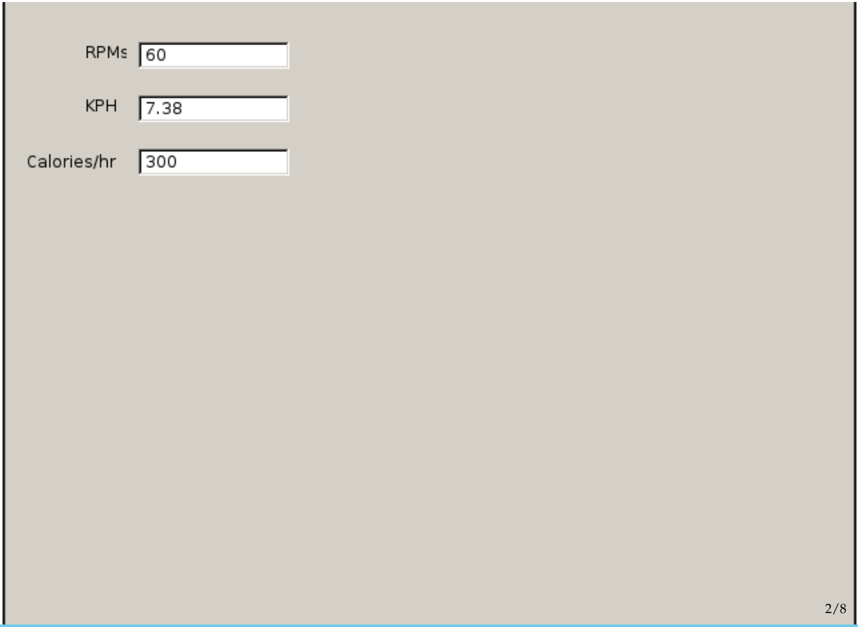
Observer Pattern

Object Oriented System Design

Otago Polytechnic
Dunedin, New Zealand

Problem: Bicycle dashboard



RPMs	60
KPH	7.38
Calories/hr	300

Problem: Bicycle dashboard

- We will enter the RPMs.
- When the RPMs change, we update
  - the speed
  - the calories per hour

Solution: Subject/Observers

Classes involved:

Bicycle (*subject*) keeps track of its RPMs

Speedometer (*observer*) determines speed from RPMs

Calorie meter (*observer*) determines calories/hour from RPMs

Implementing subject/observer

- The *subject* maintains a list of its observers.
- It *notifies* the observers when an event occurs.
- The *observers* register themselves with their subject.
- They provide an `update` method to respond to notifications from the subject.

Subject code

```
class Bicyle:

   def add_observer(self, o):
       # append o to list of observers

   def remove_observer(self, o):
       # remove o from list

   def notify_observers(self):
       # iterate over observer list and call each
       # of their update methods
```

Observer code

```
class Speedometer:

    def __init__(self, subject):
        # save reference to subject
        # call subject's add_observer method,
        # passing in self

    def update(self, rpms):
        # subject will call this when rpms change
```

Practical exercise

- Write the needed bicycle dashboard classes, `Bicycle`, `Speedometer`, `CalorieMeter` using an observer pattern.
- Use a wheel circumference of 205 cm for speed calculations.
- You can test these in your interactive Python interpreter session, but you may want to build a gui for this.
- See `http://pythonforengineers.com/your-first-gui-app-with-python-and-pyqt/`