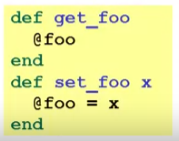
**Who can access what?**

* We know “hiding things” is essential
  + For modularity and abstraction
* OOP languages generally have various ways to hide (or not) instance variables, methods, classes, etc.
  + Ruby is no exception
* Some basic Ruby rules here as an example…

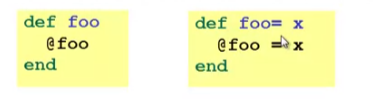
**Object state is private**

* In Ruby, object state is always private
  + Only an object’s methods can access its instance variables
  + **Not even another instance of the same class**
  + So can write @foo but not [e.@foo](mailto:e.@foo)
* To make object-state publicly visible, define “getters” / ” setters”
  + - Getter – get the contents
    - Setter – assign in an instance variable
  + Better/shorter style coming next



**Conventions and sugar**

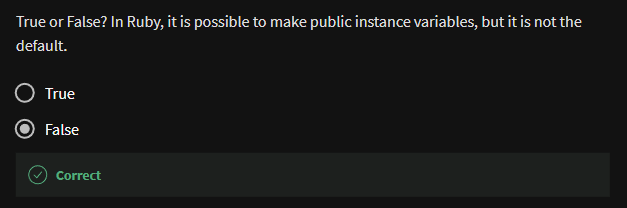
* Actually, for field @foo the convention is to name the methods:



* Cute sugar: When *using* a method ending in =, can have space before the =

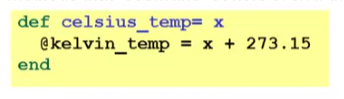


* Because defining getters/setters is so common, there is a shorthand for it
  + Define just getters: **attr\_reader :foo, :bar,** …
  + Define getters and setters: **attr\_accessor :foo, :bar,** …
* Despite sugar: getters/setters are just methods



**Why private object state**

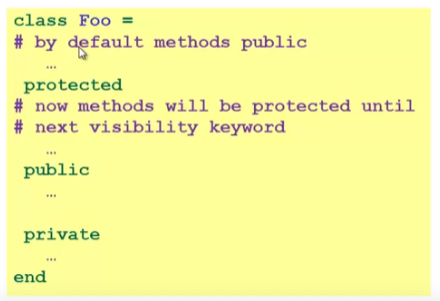
* This is “more OOP” than public instance variables
* Can later change class implementation without changing clients
  + Like we did with ML modules that hid representation
  + And like we will soon do with subclasses
* Can have methods that “seem like” setters even if they are not

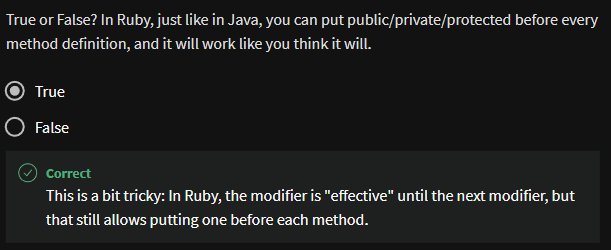


* Can have an unrelated class that implements the same methods and use it with same clients
  + See later discussion of “duck typing”

**Method visibility**

* 3 visibilities (all keywords)
  + *private*
    - only available to object itself
  + *protected*
    - available only to code in the class or subclasses
  + *public*
    - available to all code
* methods are public by default
  + multiple ways to change a method’s visibility
    - here is one way…





One detail

* If *m* is private, then you can only call it via *m* or *m(args)*
  + As usual, this is shorthand for **self.m** …
  + But for private methods, only the **shorthand** is allowed