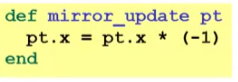
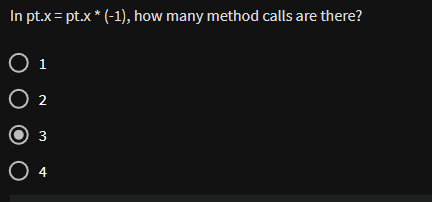
Duck typing

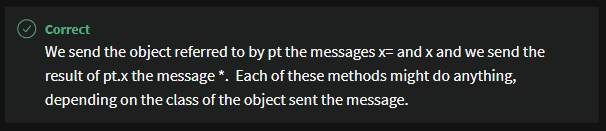
* “If it walks like a duck and quacks like a duck, it’s a duck”
  + Or don’t worry that it may not be a duck
* When writing a method you might think, “I need a Foo argument” but really you need an object with enough methods similar to Foo’s method s that your method works
  + Embracing duck typing is always making method calls rather than assuming/testing the class of arguments
* Plus: more code reuse: very OOP approach
  + What messages an object receive is “all that matters”
* Minus: almost nothing is equivalent
  + x + x versus x\*2 versus 2\*x
  + Callers may assume a lot about how callees are implemented

Example

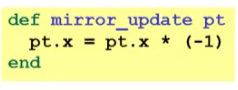


* Natural thought: “Takes a Point object (definition not shown here), negates the x value”
  + Makes sense, though a Point instance method more OOP
* Closer: “Takes anything with **getter** and **setter** methods for @x instance variable and multiplies the x field by -1”
  + There are no restrictions that **pt** must be a Point object
* Closer: “Takes anything with methods **x=** and **x** and calls **x=** with the result of multiplying result of x and -1”
  + Even if x= does not set/update any object
* Duck typing: “Takes anything with the method **x=** and **x** where result of **x** has a **\*** method that can take **-1**. Sends result of calling **x** the **\*** message with **-1** and sends that result to **x=**”
  + Downside: we already told the whole code here





With our example



* Plus: maybe **mirror\_update** is useful for classes we did not anticipate
* Minus: if someone does use (abuse?) duck typing here, then we cannot change the implementation of mirror\_update
  + For example, to – pt.x
* Better (?) example: Can pass this method a number, a string or a MyRational

