Properties And Refactoring

Money Class

Here's a money class.

```
class Money:
    def __init__(self, dollars, cents):
        self.dollars = dollars
        self.cents = cents
    # And some other methods...
```

Imagine this is released as a library, and many teams are using and relying on this class.

Refactor

One day, we decide to internally just keep track of cents.

```
class Money:
    def __init__(self, dollars, cents):
        self.total_cents = dollars * 100 + cents
    # And refactor other methods to use self.total_cents.
```

That creates a maintainability problem. Can you spot it?

The problem

Other code referencing its attributes suddenly breaks.

```
money = Money(27, 12)
message = "I have {:d} dollars and {:d} cents."
# This line breaks, because there's no longer
# dollars or cents attributes.
print(message.format(money.dollars, money.cents))
```

This could be a major impediment to changing the class interface.

The Solution

But it's not a problem in Python.

```
class Money:
    def __init__(self, dollars, cents):
        self.total_cents = dollars * 100 + cents
    # Getter and setter for dollars...
    @property
    def dollars(self):
        return self.total_cents // 100 # Integer division
    @dollars.setter
    def dollars(self, new_dollars):
        self.total_cents =
            100 * new_dollars + self.cents
```

Properties for Refactoring

```
    # And the getter and setter for cents.
    @property
    def cents(self):
        return self.total_cents % 100
    @cents.setter
    def cents(self, new_cents):
        self.total_cents =
            100 * self.dollars + new_cents
```

Java "properties"

```
// What you have to do in Java *from the beginning*
class Money {
    private int dollars;
    private int cents;
    public Money(int dollars, int cents) { // constructor
        setDollars(dollars);
        setCents(cents);
    }
    public void setDollars(int dollars) {
        this.dollars = dollars; }
    public int getDollars() { return this.dollars; }
    public void setCents(int cents) { this.cents = cents; }
    public int getCents() { return this.cents; }
}
```

In Python, we get the best of both worlds.