Super Scanner

Back to the Supermarket. We'll implement the code for a checkout system that handles pricing schemes such as "apples cost 50 cents, three apples cost $1.30".

Step 1

Let's model the various options for supermarket pricing.

Some things in supermarkets have simple prices: this can of beans costs $0.65. Other things have more complex prices. For example:

- three for a dollar (so what's the price if I buy 4, or 5?)
- $1.99/pound (so what does 4 ounces cost?)
- buy two, get one free (so does the third item have a price?)

Bonus

To make it better, you can consider things like:

- Start Date / End Date of an event
- How to keep an audit trail of pricing decisions?

Step 2

We'll have to implement the code for a supermarket checkout

that calculates the total price of a number of items. In a normal supermarket, things are identified using Stock Keeping Units, or SKUs. In our store, we'll use individual letters of the alphabet (`A` , `B` , `C` , ...). Our goods are priced individually. In addition, some items are multipriced: buy n of them, and they'll cost you y cents. For example, item `A` might cost 50 cents individually, but this week we have a special offer: buy three `A` s and they'll cost you `$1.30` .

```
Item    Unit Price  Special Price
  A          50         3 for 130
  B          30         2 for 45
  C          20
  D          15
```

Our checkout accepts items in any order, so that if we scan a `B` , an `A` , and another `B` , we'll recognize the two `B` 's and price them at `45` (for a total price so far of `95`). Because the pricing changes frequently, we need to be able to pass in a set of pricing rules each time we start handling a checkout transaction.

The interface to the checkout should look like:

```
co = CheckOut.new(pricing_rules)
co.scan(item)
co.scan(item)
...
price = co.total
```