ORM Modeling Problem

Test your understanding of object-relational modeling basics in Django.

Write Python code to define Django model classes for Sale, LineItem, and Product. Write only the attribute declarations, including declaration of foreign key relationships needed to save the relationship between objects. Don't write any methods.

Problem Description

A Sale consists of a collection of Line Items. Each Line Item refers to the purchase of some units (quantity) of a product, and a Product has a product code, description, and unit price. Since Line Item can compute it's price from the quantity and product price, there is no "price" attribute in Line Item. Similarly, a Sale can compute the total price just by summing the total price of line items, so there is no "total" attribute in Sale. However, for accounting reasons the Sale *does* record the amount of tax added.

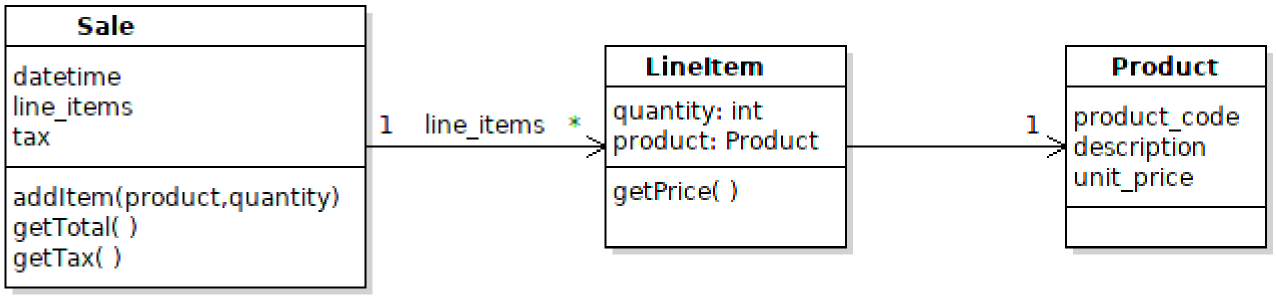
All money values should be stored in **decimal format** with 2 decimals (like Thai money).

Product descriptions are always 40 characters or less. (If an actual product description is longer, we abbreviate it).

The product codes may be UPC-A (12 digits), UPC-E (6 digits), EAN-8 (8-digits), EAN-13 (13 digits), or the store's own wacky, nonstandard Stock Keeping Unit (SKU) code (10 digits) for things that don't have a UPC/EAN code. Some codes begin with zeros, which must be preserved, so don't convert 000000123456 to 123456.

Every product in the database should have a unique product code. Our database guru recommends that you do *not* use product code as the primary key; let Django manage the primary key itself.

For info about product codes see the Wikipedia page for "Universal Product Code".



Problem

How would you define the Sale, LineItem, and Product classes in Django?

Write only the class declaration and the fields (attributes). Use the most appropriate field type, field size, and include any useful constraints on the field values (things that would help keep bad data out of the database). Include relationship between objects. Use the Python package "`sales`" for your code.

What to Submit

This is optional for learning. If you do it I will check our solution and comment on Github.

Submit your code using this Github assignment: https://classroom.github.com/a/9zOx8tbj

Django Validators

Code that validates field values for Django models are called "validators". Validators are applied when you call form.is_valid() for HTML form data, or if you call fieldname.validate() for any field in a Model.

Django has many built-in validators and you can write your own as a function with one parameter.

Validators save you a lot of code, since they will validate data from forms and generate error messages. If you use the Django Form views, it adds some client-side validation (but not for all validators), too.

Info about Validators: https://docs.djangoproject.com/en/3.1/ref/validators/

Suppose that a Person's name should always contain at least 3 characters. You could write:

```
from django.core.validators import MinLengthValidator

class Person(models.Model):
    name = models.CharField(max_length=80,
                validators=[MinLengthValidator(3)], blank=False)
```

The validators parameter is a list of validators, so you can use more than one.
Some validators are applied automatically. An EmailValidator is used for any EmailField, but you might specify a custom EmailValidator to only allow domains "ku.th" or "ku.ac.th".