Programming Project B

Comprehensions

In the Canvas folder for this assignment (Files/Projects/ProjB) are text files named *parts.txt*, *projects.txt*, *spj.txt*, and *suppliers.txt* that contain comma-separated data for the 4 database tables seen in class: **parts**, **projects**, **spj**, and **suppliers**. As a reminder, the "data dictionary" (slot description) is:

- **suppliers:** (sno,sname,status,city)
- **parts:** (pno,pname,color,weight,city)
- **projects:** (jno,jname,city)
- **spj:** (sno,pno,jno,qty)

The first slot in each tuple (**sno**, **pno**, and **jno**) in the first 3 tables is the unique record id (a string). You may let all of your slot data types be strings for simplicity in this assignment.

Each text file has the name of the relation/table and the slot/column names on the first two lines, respectively. Use these to properly define your named tuples.

For this project, read the data from each file into an associated set of **namedtuples**, so you can process the queries below using **set and dictionary comprehensions** (a dictionary comprehension for #6).

To illustrate, you can find suppliers for projects in London as shown below in the file *london_named.py* in the Code folder:

```
{s.sname for s in suppliers for r in spj for j in projects
 if r.sno == s.sno if j.jno == r.jno if j.city == 'London'}
```

which gives the response:

```
{'Jones', 'Adams', 'Clark'}
```

We use sets so that any duplicate tuples are removed, as is typical in database queries. For readability, you could separate the queries into multiple steps, as found in *london_named.py*, shown below:

```
london_projs = {j.jno for j in projects if j.city == 'London'}
london_supp_ids = {r.sno for r in spj if r.jno in london_projs}
london_supps = {s.sname for s in suppliers if s.sno in london_supp_ids}
print(london_supps)
```

You can decide which form looks best to you for each problem individually.

Each text file has the name of the relation/table and the slot/column names on the first two lines, respectively. Use these to properly define your named tuples. To illustrate, here is *parts.txt*:

```
parts
pno,pname,color,weight,city
p1,Nut,Red,12,London
p2,Bolt,Green,17,Paris
p3,Screw,Blue,17,Rome
p4,Screw,Red,14,London
p5,Cam,Blue,12,Paris
p6,Cog,Red,19,London
```

The code that reads such data should be application independent/generic. I would write a function that returns the named tuple object. (*Hint*: Consider using the unary * sequence-flattening operator for arguments in a function call.)

Write code that uses the named tuples mentioned above for the following queries:

1. Get names of all suppliers that supply bolts.
2. Get names of all suppliers that supply blue parts.
3. Get names of all suppliers not used in Athens projects
4. Get names and colors of all parts not used in Oslo
5. Get pairs of names of all suppliers that are located in the same city.
6. Print all suppliers out by city

Print no duplicates. You should also have *no top-level loops* in this script (they are inside the comprehensions) except for when you read the data in. The queries should use **comprehensions only**.

Here is the expected output (order within a set or dictionary is immaterial, for this project, and also within the pairs of suppliers in the same city):

```
{'Adams'}
{'Jones', 'Adams', 'Blake'}
{'Blake'}
{('Cam', 'Blue'), ('Cog', 'Red'), ('Nut', 'Red'), ('Bolt', 'Green'), ('Screw', 'Red')}
{('Jones', 'Blake'), ('Smith', 'Clark')}
{'Paris': {'Jones', 'Blake'}, 'Athens': {'Adams'}, 'London': {'Clark', 'Smith'}}
```

Use the following **pylint** directive:

```
# pylint: disable=invalid-name, undefined-loop-variable
```

Submit your code in a file named *comp.py*.