Problem 1

Write a class Room, which has the following properties:
- Building Name
- Room Number
- Size (2 dimensions)
- Maximum Occupancy (max # of people who can be inside)
- Owning Department Name

Write an __init__() method. Include at least 3 assert statements to validate the various parameters.

Suppose that you had a pair of dictionaries, which listed the names of all the buildings, and all of the departments. Use the “in” expression to confirm that the building and department names are valid names. (If you have forgotten this expression, do a Google search for “python in dictionary”.)

While in a real program, we would have getters for all of the properties, skip over those methods this time.

Thinking about setters: which of the properties are things which might change over time? We would need setters for those properties. (But it's kind of pointless to write a setter for something that will never change.) Write down which properties you think might change, and which might not. Explain your reasoning.
Problem 2

In Problem 1, we wrote a Room class - which represents a few properties about a Room which would either never change, or change very rarely. But there are other properties, which might change more rapidly.

One option is to simply add more fields to Room - and that's not a terrible idea. But let's explore another option: a RoomState class. Each RoomState class will contain a reference to a single Room object (this reference will never change), and many other fields which will change often.

Define a RoomState class. The __init__() method must take a reference to the Room object - and this reference must never change after __init__() finishes. Define at least 4 other properties that you might store about a room. For at least one of them, pass the initial state as a parameter to __init__() - for at least one of them, do NOT use a parameter, and instead give the parameter a default value. (You get to define what the various other properties are.)

Skip getters for these properties, to save time. However, write at least 4 different setter methods (they could all be for the same property, or for different ones). But here's the trick: none of the setters are allowed to simply take “the new value” as a parameter. Instead, they must change the property indirectly. For instance, if you a temperature property, then you might have a become_hotter() method.

Then, write a single method, which answers a question about the class, but is more interesting than a simple getter. For instance, you could as is_dangerously_hot(), or is_occupancy_exceeded().
Problem 3

I have written some if statements below. At the beginning of each block, write all of the assertions which you know are true. (Assume that each “condA” represents some sort of expression with a Boolean result - that is, it is either True or False.)

```python
assert condA
if condB:
    # example
    assert condA
    assert condB
    return []

if condC:
    # write assertions here

elif condD:
    # write assertions here

globally:
    # write assertions here
else:
    # write assertions here
```