Problem

Our bioprocess instructions typically dictate different strategies to pump glucose into the vessel to achieve the desired cell growth and metabolism. Implement a cascaded feed/pump control interface in Python that can be used by recipes to feed cells. Also, please provide the rough time it took to complete.

Details

We refer to the container holding the organism and media that it grows in the “vessel”. Liquids are typically pumped into this (and sometimes out of it) from other small containers nearby.

We have two relevant sensors:

- **Feed Scale**: The container that the glucose is in sits on a scale. As the glucose is pumped into the fermentation vessel, the scale reading will change. We can read this value in grams, and it is always highly reliable.
- **Feed Pump Steps**: The volume of liquid the pump has delivered is reported in a discrete quantity of “steps”.

You can assume there is some constant parameter that describes the average volume of liquid delivered with each “step”. However, this is only an average and the actual amount delivered will vary.

You can assume you have some interface to read these sensor values, maybe like this:

```python
import enum



class Sensor(enum.Enum):
   FEED_SCALE_G = "feed_scale"
   FEED_PUMP_STEPS = "feed_pump_steps"



def get_sensor_value(sensor: Sensor) -> float:
   # Pretend this works
   pass
```

Sensor values are always reliable.

To control the pump, you can set:

- **Feed Pump Step Rate**: in steps/min, how fast the pump will try to deliver liquid. This must always be set.
- **Feed Pump Step Target**: in number of steps, the pump will stop after it reaches this many steps. This is optional, if not given, the pump will operate until it is stopped.

You can assume you have some interface to set these values, maybe like this:

```
class Actuator(enum.Enum):
   FEED_PUMP_STEP_RATE = "feed_pump_step_rate"
   FEED_PUMP_STEP_TARGET = "feed_pump_step_target"


def set_actuator_value(actuator: Actuator, value: float) -> None:
   # Pretend this works
   pass
```

We call the bioprocess instructions that are specific to each customer's experiment the "recipe". The recipe has different ways of dictating how to feed glucose:

- **Bolus**: feed in a target mass of glucose
- **Linear**: feed in glucose at a steadily increasing or decreasing volumetric rate up to a max/min
- **Timed**: feed in glucose at a given volumetric rate for a given time

The recipe will only use one of these at a time, but might use e.g. one then another. *Try to implement all of these.*

Make up any parameters or functions that you need.

Example Recipe (this is the general idea, but feel free to come up with your own interface):

```
import recipe_interface


# Recipes will set up the experiment in different ways.
# Don't worry about this stuff
recipe_interface.set_up_experiment()

# An example of what the recipe would do - start feeding glucose.
# Your solution would include an implementation of this function.
```

```
# As written here, we assume it doesn't block but only
# starts the feeding and the recipe would have to check
# later to see if it's done.  You might want to weigh pros/cons
# of blocking vs nonblocking.
recipe_interface.start_bolus_feed(<params>)

# This is from the nonblocking nature of the above call,
# but feel free to reason about advantages/disadvantages.
while not recipe_interface.is_feeding_done():
   time.sleep(10)

# Something like this should always be present.
recipe_interface.shutdown()
```