Elevator simulator

Author: Nat Kerman

Date: 2022-06-06

Description of the task

As per email from Wendy Carande *(with key terms highlighted by Nat)*:

> Construct a **class** that simulates an elevator that is on the **Xth floor of a Z story building**. The elevator travels at 1 floor/10 seconds (don't worry about acceleration/deceleration, etc.). **Input** will include a list of floor requests. **Output** will be: Current floor of the elevator, floors left to arrive at destination, time left to arrive at destination. Be sure to **list any assumptions** you feel should be known. The language preference is Python. Keep in mind we will want to run it, and we are interested in seeing software engineering best practices including **documentation, readability, testing**, etc.

Assumptions made

1. Maximum height of the elevator is the top floor (Z) of the building.
2. There is no minumum floor; floor 0 is ground and there are as many basement floors as the inputs demand.
3. All floors are integers. Perhaps a future version of this code could support mezzanine levels as fractional floors; however, it seems a reasonable assumption that all floors can be expected to be integers.
4. There is no acceleration/deceleration.
5. No time is spent at each stop.
6. Python version ≥ 3.6 (I will use f-strings).
7. If a user inputs an invalid value, the desired behavior is a failure with a clear assertion error.
8. Given the somewhat unclear wording of "outputs" in the prompt (an Elevator *class* was requested, and classes have states while methods and functions have outputs), I have opted to store the desired outputs in the state of the object and include a human-readable version in the `__repr__` of the object. This `__repr__` is optionally printed at each floor, and the history of floors, time elapsed, and time to destination is always kept in state. If a specific *output* were requested, I could happily build a method to output the current floor, time to next floor, etc. for a given state change. In this case, it did not seem the best way to accomplish the goals, and the wording of the task did not seem picky. In a non-test setting, I would discuss this with my client/supervisor and determine the best implementation solution for the task at hand.

How to use the Elevator class

As requested, the code is built around an `Elevator` class, which holds key information about its location, history, and future in its state. Please see the file `examples.py`, which contains specific examples of using the Elevator object, but it's also illustrated briefly here.

```
from elevator import Elevator
# Instantiates an Elevator
el = Elevator(X=0, Z=100, name="The Great Glass Elevator", building_name="The Chocolate Factory")
# Travel through a list of floors
el.travel_through_floor_list([1,20,3,9,100], verbosity=2)
```

To see all the information and methods contained in an Elevator object, create the Elevator object el as above, then run the following:

```
dir(el)
```

Testing the Elevator object code

I have built 9 tests into `test_elevator.py`. To run them...

1. install `pytest`
2. `cd` into the directory with `elevator.py` and `test_elevator.py`
3. from the shell, run `pytest -v`

Requirements

1. `Python` ≥ 3.6
2. `numpy`
3. `pytest` for running tests in `test_elevator.py`

- Note that if your version of `pytest` is very old, the tests will still pass but it may return warnings about `distutils` deprecation. These are about a dependency of `pytest`, and don't have anything to do with this code. Updating your environment packages should fix this.