Object-Oriented Programming

A productive approach to defining new classes is to determine what instance attributes each object should have and what class attributes each class should have. First, describe the type of each attribute and how it will be used, then try to implement the class's methods in terms of those attributes.

Q1: Keyboard

Overview: A keyboard has a button for every letter of the alphabet. When a button is pressed, it outputs its letter by calling an `output` function (such as `print`). Whether that letter is uppercase or lowercase depends on how many times the *caps lock* key has been pressed.

First, implement the `Button` class, which takes a lowercase `letter` (a string) and a one-argument `output` function, such as `Button('c', print)`.

The `press` method of a `Button` calls its `output` attribute (a function) on its `letter` attribute: either uppercase if `caps_lock` has been pressed an odd number of times or lowercase otherwise. The `press` method also increments `pressed` and returns the key that was pressed. *Hint*: `'hi'.upper()` evaluates to `'HI'`.

Second, implement the `Keyboard` class. A `Keyboard` has a dictionary called `keys` containing a `Button` (with its `letter` as its key) for each letter in `LOWERCASE_LETTERS`. It also has a list of the letters `typed`, which may be a mix of uppercase and lowercase letters.

The `type` method takes a string `word` containing only lowercase letters. It invokes the `press` method of the `Button` in `keys` for each letter in `word`, which adds a letter (either lowercase or uppercase depending on `caps_lock`) to the `Keyboard`'s `typed` list. **Important:** Do not use `upper` or `letter` in your implementation of `type`; just call `press` instead.

Read the doctests and talk about:

- Why it's possible to press a button repeatedly with `.press().press().press()`.
- Why pressing a button repeatedly sometimes prints on only one line and sometimes prints multiple lines.
- Why `bored.typed` has 10 elements at the end.

```
LOWERCASE_LETTERS = 'abcdefghijklmnopqrstuvwxyz'

class CapsLock:
    def __init__(self):
        self.pressed = 0

    def press(self):
        self.pressed += 1

class Button:
    """A button on a keyboard.

    >>> f = lambda c: print(c, end='')  # The end='' argument avoids going to a new line
    >>> k, e, y = Button('k', f), Button('e', f), Button('y', f)
    >>> s = e.press().press().press()
    eee
    >>> caps = Button.caps_lock
    >>> t = [x.press() for x in [k, e, y, caps, e, e, k, caps, e, y, e, caps, y, e, e]]
    keyEEKeyeYEE
    >>> u = Button('a', print).press().press().press()
    A
    A
    A
    """
    caps_lock = CapsLock()

    def __init__(self, letter, output):
        assert letter in LOWERCASE_LETTERS
        self.letter = letter
        self.output = output
        self.pressed = 0

    def press(self):
        """Call output on letter (maybe uppercased), then return the button that was
    pressed."""
        self.pressed += 1
        "*** YOUR CODE HERE ***"
```

```
class Keyboard:
    """A keyboard.

    >>> Button.caps_lock.pressed = 0  # Reset the caps_lock key
    >>> bored = Keyboard()
    >>> bored.type('hello')
    >>> bored.typed
    ['h', 'e', 'l', 'l', 'o']
    >>> bored.keys['l'].pressed
    2

    >>> Button.caps_lock.press()
    >>> bored.type('hello')
    >>> bored.typed
    ['h', 'e', 'l', 'l', 'o', 'H', 'E', 'L', 'L', 'O']
    >>> bored.keys['l'].pressed
    4
    """
    def __init__(self):
        self.typed = []
        self.keys = ...  # Try a dictionary comprehension!

    def type(self, word):
        """Press the button for each letter in word."""
        assert all([w in LOWERCASE_LETTERS for w in word]), 'word must be all lowercase'
        "*** YOUR CODE HERE ***"
```

Description Time: Describe how new letters are added to `typed` each time a `Button` in `keys` is pressed. Instead of just reading your code, say what it does (e.g., "When the button of a keyboard is pressed …"). One short sentence is enough to describe how new letters are added to `typed`.