My fill() function comprises three steps:

1. Call the validate_seed_point() function which returns True if the seed_point given is valid, False otherwise. If it's invalid, fill() simply returns the original image.
    - validate_seed_point() checks the tuple seed_point. First, it checks whether both elements of the tuple are integers. If not, return False. Second, it confirms whether both elements lie within the image's grid range. If either doesn't, it returns False. Finally, it checks whether the seed_point begins on a boundary (i.e. a value of 1) If so, it returns False. In each case, the relevant error message should be printed to the console.

2. If the tests for validate_seed_point() pass, fill() next calls the fill_surrounding_pixels() function, which in turn is comprised of two ancillary functions.
    - get_bordering_pixels() looks at the pixels adjacent to the current co-ordinates i.e. the pixels left, right, up and down. For each, it defines the bordering pixels' co-ordinates (set to None if they fall outside the image bounds). They also retrieve the pixels' value so we can check whether it needs to be filled in. This returns a nested dictionary for each pixel, giving the co-ordinates and values of its surrounding pixels.
    - fill_surrounding_pixels() is a recursive function at the heart of the filling process. From a starting point, it checks if the value is unfilled (recall we should always start on an unfilled cell from validate_seed_point()). If it is unfilled, we fill it. Next, we get the positions and values of adjacent pixels. We loop through each adjacent pixel and, if it's unfilled, we move to it and begin the process again by recursion.

3. Return the image once filled.

My battery of tests can be found in the second script.