Daily Coding Problem #206

Problem

This problem was asked by Twitter.

A permutation can be specified by an array `P`, where `P[i]` represents the location of the element at `i` in the permutation. For example, `[2, 1, 0]` represents the permutation where elements at the index `0` and `2` are swapped.

Given an array and a permutation, apply the permutation to the array. For example, given the array `["a", "b", "c"]` and the permutation `[2, 1, 0]`, return `["c", "b", "a"]`.

Solution

If there are no restrictions on memory, we can do this in a straightforward way in `O(N)` time by creating a new list:

```
def permute(array, permutation):
    return [array[p] for p in permutation]
```

Solving this in place requires some more thought. To gain some intuition, let's look at an example: applying `[3, 0, 2, 1]` to `["a", "b", "c", "d"]`.

We can apply this permutation using three swaps: moving a to where d is, d to where b is, and b to where a is. Essentially what we did was cycle the elements from their original location to their permuted indices until coming back to our start index. When our cycle finished (`a -> d -> b -> a`), we went to the next un-permuted element, c, and applied the logic again, except that in this case the permutation array did not require us to make any moves.

To implement this, we will loop through each element of the array, and if the element is not in the right place, initiate a cycle of swaps. While we have not yet come back to our start index, we move elements to their permutation index and update the permutation array so we know not to permute that element again. Finally, to complete each cycle we will need to remember the last-replaced element and last-permuted index, for which we will use temp variables `element` and `p`.

```
def permute(array, permutation):
    for i in range(len(array)):
        element, p = array[i], permutation[i]

        while p != i:
            array[p], element = element, array[p]
            permutation[p], p = p, permutation[p]

        array[i], permutation[i] = element, p
    return array
```

Although it seems we are looping over the elements twice, note that we only enter the inner loop when the array element needs to be permuted. Since we only end up moving each element once, this algorithm is still `O(N)`.