1. Recall our function `collect_underperformers`. Assume this function is in a module named `underperformers.py`.

```python
def collect_underperformers(nums: List[int], threshold: int) -> List[int]:
    """Return a new list consisting of those numbers in nums that are below threshold,
in the same order as in nums.
    """
```

(a) We’ve begun writing a test suite for this function using `unittest`. Complete methods `test_underperformers_high_threshold` and `test_underperformers_mutation`.

```python
import unittest
import underperformers

class TestCollectUnderperformers(unittest.TestCase):

    def test_underperformers_low_threshold(self):
        """Test collect_underperformers with a threshold for which there
        are no underperformers.
        """

        actual_underperformers = underperformers.collect_underperformers([4, 5, 6], 1)
        expected_underperformers = []
        self.assertEqual(actual_underperformers, expected_underperformers)

    def test_underperformers_high_threshold(self):
        """Test collect_underperformers with a threshold for which all items
        are underperformers.
        """

    def test_underperformers_mutation(self):
        """Confirm that collect_underperformers does not mutate the list it’s given.
        """
```

Note: the test suite above is not complete!
(b) We’ve changed our mind about the desired behaviour for `collect_underperformers`. We’d like it to modify the list it is given, and not return anything. Consider the new function below:

```python
def keep_underperformers(nums: List[int], threshold: int) -> None:
    """Modify nums to only contain those numbers that are
    below threshold, in the same order as in nums.
    """
```

Rewrite the first testing method above to work with `keep_underperformers`'s new description.

```python
class TestKeepUnderperformers(unittest.TestCase):
    def test_underperformers_low_threshold(self):
        """Test keep_underperformers with a threshold for which there
        are no underperformers.
        """
```

2. Complete the two test methods for `most_popular`, described below.

```python
def most_popular(company_to_placements: Dict[str, List[int]]) -> List[str]:
    """Return the company (or companies) with the most placements in the race
    according to company_to_placements.
    
    Precondition: company_to_placements is not empty
    """
```

```python
class TestMostPopular(unittest.TestCase):
    def test_most_popular_one_item(self):
        """Test most_popular with a dictionary of length 1.
        """
```

```python
def test_most_popular_mutation(self):
    """Confirm that most_popular does not mutate the dict it is given.
    """
```