CSC108H Winter 2024 Worksheet 23 : check_password

For the function `check_password`, complete the table below with test cases. Test only with valid input and avoid duplicate tests.

```
def check_password(passwd: str) -> bool:
    """A strong password has a length greater than or equal to 6, contains at
    least one lowercase letter, at least one uppercase letter, and at least
    one digit.  Return True if and only if passwd is considered strong.

    >>> check_password('I<3csc108')
    True
    """
```

Test Case Description	passwd	Expected (True or False)
Meets all requirements	'Hell00'	True
Exactly 6 characters with all requirements	'Helll0'	True
More than 6 characters with all requirements	'Hell000'	True
No uppercase letter(s)	'hell000'	True
No lowercase letter(s)	'H123456'	False
No digit(s)	'Abcdef'	False
Empty string	''	False
Less than 6 characters with all requirements	'Ab0'	False

CSC108H Winter 2024 Worksheet 24 : Choosing Test Cases

For each of the following functions, choose a set of test cases. Test only with valid input and avoid duplicate tests. **The tables may contain more rows than necessary.** Leave the **Actual** and **Pass?** columns blank for now.
After you have completed your test cases, you will receive buggy code to see how well your tests worked. That is what the **Actual** and **Pass?** columns are for.

1.
```
def is_teenager(age: int) -> bool:
    """Return True if and only if age is a teenager between 13 and 18 inclusive.

    Precondition: age >= 0
    """
```

Test Case Description	`age`	**Expected**	**Actual**	**Pass?**

2.
```
def all_fluffy(s: str) -> bool:
    """Return True if and only if every character in s is fluffy. Fluffy
    characters are those that appear in the word 'fluffy'.
    """
```

Test Case Description	`s`	**Expected**	**Actual**	**Pass?**

CSC108H Winter 2024 Worksheet 24 : Choosing Test Cases

3.
```
def same_abs(int1: int, int2: int) -> bool:
    """Return True if and only if int1 and int2 have the same absolute value."""
```

Test Case Description	int1	int2	Expected	Actual	Pass?

4.
```
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.

    If one company has the most placements, the returned list contains only that company.
    If there is a tie, the returned list is sorted alphabetically.

    Precondition: company_to_placements is not empty
    """
```

Test Case Description	company_to_placements	Expected	Actual	Pass?

CSC108H Winter 2024 Worksheet 25 : pytest

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

```
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 `pytest`. Complete functions `test_underperformers_high_threshold` and `test_underperformers_mutation`.

```
import pytest
from underperformers import collect_underperformers

def test_low_threshold() -> None:
    """Test collect_underperformers with a threshold for which there
    are no underperformers.
    """

    actual_underperformers = collect_underperformers([4, 5, 6], 1)
    expected_underperformers = []
    assert actual_underperformers == expected_underperformers

def test_high_threshold() -> None:
    """Test collect_underperformers with a threshold for which all items
    are underperformers.
    """
```

```
def test_mutation() -> None:
    """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 outcome for function `collect_underperformers`. We would instead like to have a function that modifies the given list and does not return anything. Consider the function `keep_underperformers` that is also defined in the `underperformers.py` module:

```
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 function above to work with `keep_underperformers`'s description.

```
from underperformers import keep_underperformers

def test_low_threshold() -> None:
    """Test keep_underperformers with a threshold for which there
    are no underperformers.
    """
```

2. Complete the two test functions for `most_popular`, described below. Assume this function is in a module named `running.py`.

```
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.

    If one company has the most placements, the returned list contains only that company.
    If there is a tie, the returned list is sorted alphabetically.

    Precondition: company_to_placements is not empty
    """
```

```
from running import most_popular

def test_one_item() -> None:
    """Test most_popular with a dictionary of length 1.
    """
```

```
def test_mutation() -> None:
    """Confirm that most_popular does not mutate the dict it is given.
    """
```

CSC108H Winter 2024 Worksheet 26 : Testing Functions that Return Floats

Consider the function `get_average_item_price`.

```
def get_average_item_price(expenses: dict[str, list[float]]) -> float:
    """Return the average price of all the items in expenses. expenses is a
    dictionary of people to lists of the prices of items each person purchased.
    """
```

1. We have written two tests for the function, but only one of them passes.

```
def test_one_person_one_item() -> None:
    """Test one person with one expense.
    """
    actual = get_average_item_price({'Tom': [5.0]})
    expected = 5.0
    assert actual == expected
```

```
def test_one_person_many_items() -> None:
    """Test one person with many items.
    """
    actual = get_average_item_price({'Tom': [14.99, 3.25, 5.0]})
    expected = 7.7466666666666666
    assert actual == expected
```

   (a) Assume you've carefully stepped through the code using the debugger, and you have confidence it is correct. The fact that one test case is still failing leads you to suspect there is likely a problem with the test itself. Which test case do you think is most likely to fail? (If you run this code yourself, the tests might pass – but they won't for everyone!)

   (b) Fix any issues with the tests above. (Hint: recall a useful `pytest` function!)

2. Complete the following test case for `get_average_item_price`.

```
def test_many_people_with_one_item() -> None:
    """Test many people each with only one item.
    """
```

Note: the test suite above is not complete!