CSE 2050: Homework 03

For each question in this homework, you are tasked with implementing a function that:

- exhibits the correct behavior, and
- meets the required running time complexity.

You cannot receive full credit on a problem unless BOTH conditions are met.

Problem 1 - `remove_characters.py`

Write a function `remove_characters(input_string, to_remove)` that takes two inputs:

- `input_string` : a string
- `to_remove`: a string

and returns a string that is exactly the same as the `input_string` but without any of the characters in `to_remove`. Your function must have a running time complexity of $O(n)$, where n represents the length of `input_string`.

For clarity, let m be the length of `to_remove`. Your code cannot have a running time complexity of $O(mn)$ to receive full credit, it must be $O(n)$.

NOTE: you may not use any built-in string removal methods/functions in your code (e.g., `str.strip()`).

Examples

These examples are intended to be illustrative, not exhaustive. Your code may have bugs even if it behaves as below. Write your own unittests to further test expected behaviors.

```
>>> new_string = remove_characters('abcd', 'c')
>>> print(new_string)
'abd'
```

Problem 2 - `any_two_sum.py`

Write a function `any_two_sum` that takes two inputs:

- `numbers` : a list of integers
- `total`: an integer

and returns `True` if any two integers in `numbers` sum to `total` and `False` otherwise. Your function must have a running time complexity of $O(n)$, where n represents the length of `numbers`.

Examples

These examples are intended to be illustrative, not exhaustive. Your code may have bugs even if it behaves as below. Write your own unittests to further test expected behaviors.

```
>>> result = any_two_sum([1,3,4,5], 7)
>>> print(result)
True
```

```
>>> result = any_two_sum([1,3,4,5], 2)
>>> print(result)
False
```

Problem 3 - `contains_permutation.py`

Write a function `contains_permutation` that takes two inputs:

- `input_string` : a string
- `pattern`: a string

and returns `True` if `input_string` contains a substring (group of consequtive characters) that is a permutation of `pattern` and `False` otherwise. Your function must have a running time complexity of $O(n)$, where n is the length of `input_string`.

For clarity, a permuation of `pattern` would be a string of equal length that use the exact same characters, but in any order. For example, `'abc'` is a permutation of `'cab'`.

Examples

These examples are intended to be illustrative, not exhaustive. Your code may have bugs even if it behaves as below. Write your own unittests to further test expected behaviors.

```
>>> result = contains_permutation('abcdef', 'cab')
>>> print(result)
True
```

```
>>> result = contains_permutation('keyboard', 'boy')
>>> print(result)
True
```

```
>>> result = contains_permutation('patriots', 'sit')
>>> print(result)
False
```

Imports

We do not allow imports on any homework or lab assignments, except for those explicitly specified by that assignment. Due to the ubiquitous nature of the problems we are solving, there are often modules that trivialize the assignments when imported, so we restrict imports to ensure you're learning all relevant techniques.

Do not import any modules except for those you write yourself, or any exceptions below:

- You can import the `unittest` and `random` modules for testing purposes.
- You may import `typing` – not required, but some students have requested this module.

Grading

Some of the functionality will be auto-graded. However, we will manually grade for structure, readability, efficiency, and testing - ensure that:

- Every function has a docstring (more info)

- Code is well commented - you don't need a comment on every line, but don't assume because *you* know what something does that a stranger (or you in two months) will understand it.
- You use consistent variable naming conventions (see the Python style guide)
- Required running time complexities are met.
- Every function has a set of comprehensive unit tests (submitted in a separate file as shown below).

Submitting

Submit the following files:

- `remove_characters.py`
- `test_remove_characters.py`
- `any_two_sum.py`
- `test_any_two_sum.py`
- `contains_permutation.py`
- `test_contains_permutation.py`

Students must submit to Gradescope individually by the due date (typically Tuesday at 11:59 pm EST) to receive credit.

Optional Challenge Problem - `minimum_substring.py`

Write a function `minimum_substring` that takes two inputs:

- `input_string` : a string
- `to_match`: a string

and returns the smallest substring (group of consequtive characters) of `input_string` that contains all of characters within `to_match`, if it exists. Otherwise, your function should return `None`. You function must have a running time complexity of $O(n)$.

Examples

These examples are intended to be illustrative, not exhaustive. Your code may have bugs even if it behaves as below. Write your own unittests to further test expected behaviors.

```
>>> result = minimum_substring('abcdef', 'cab')
>>> print(result)
'abc'
```

```
>>> result = minimum_substring('keyboard', 'boy')
>>> print(result)
'ybo'
```

```
>>> result = minimum_substring('patriots', 'sit')
>>> print(result)
'iots'
```