Python – Lab exercise 6

Python allows you to specify the stride when slicing a string. For instance

```
>>> s = "Python is Awesome"
>>> s[0:16:2]
'Pto sAeo'
>>> s[1:16:2]
'yhni wsm'
```

Also, python allows you to specify negative numbers as strides. Experiment to see what a negative number as a stride would do.

1. Now using the above observation (negative strides) to reverse a string, implement the function specified in Q4 of Lab exercise 5. (repeated below)

   Write a function that accepts a string as input parameter and prints whether it is a palindrome or not. Ignore the case of the characters. Write the specification including doctests (test cases in docstring) for the function.

2. Recall problem 3 from Lab exercise 5 (repeated below).

   You are required to write the following function:

   `unique_vowels_in_word(s, case_sensitive=False)`

   that counts the number of unique vowels (a,e,i,o,u) in a given string s. Upper-case 'A' and lower-case 'a' should be considered as the same if case_sensitive is False (default case) and different otherwise. The function should return -1 if the string s contains any non-letter. Write the specification including doctests (test cases in docstring) for the function.

   Implement the above function and copy paste your code below. Run it and see if it passes all the doctests written by you. Now run using pytest to see if it still passes all the test cases. Were your test cases good enough?

```
def unique_vowels_in_word(s, case_sensitive=False):
    # copy-paste your code here

### DO NOT MODIFY BELOW THIS LINE ###

# Model solution (DO NOT simply copy-paste this above!)
def sol_unique_vowels_in_word(s, case_sensitive=False):
    if not s.isalpha():
    return -1
    vowels = 'aeiouAEIOU'
    if not case_sensitive:
    s = s.lower()
    return len(set(s).intersection(vowels))

# Use Hypothesis
from hypothesis import given, strategies as st, settings

# Run plenty of examples!
@settings(max_examples=1000)

# Test with default argument (False)
@given(st.text())
def test_1_arg(s):
    assert unique_vowels_in_word(s) == sol_unique_vowels_in_word(s)
```

```
# Test with two arguments
@given(st.text(), st.booleans())
def test_2_arg(s, b):
    assert unique_vowels_in_word(s, b) == sol_unique_vowels_in_word(s, b)


# Run the Hypothesis
if __name__ == '__main__':
    test_1_arg()
    test_2_arg()
```

3. Implement problem 2 from Lab exercise 5 (repeated below) and run it to see if it passes all the doctests written by you. After you are done with your implementation, a model solution would be provided for you to perform hypothesis testing.

   You are required to write a function that receives a string **s** as input parameter and returns **True** if the vowels in **s** are in alphabetical order, and returns **False** otherwise. In other words, the first occurrence of '**a**' (if any) appears before first **e** (if any), first '**e**' appears before first '**i**' if any etc. The signature of the function is as follows:

   vowels_alphabetical(s)

   Write the specification including doctests (test cases in docstring) for the function.

4. Implement problem 1 from Lab exercise 5 (repeated below) and run it to see if it passes all the doctests written by you.

   You are required to write a function that receives as input parameters the times for two events (such as 3pm and 11am), determines whether the first event occurs before the second event, at the same time, or after the second event, and prints "Before", "Same", or "After" accordingly. The signature of the function is given below:

   find_chronology(time1, suffix1, time2, suffix2)

Here, suffix is “am” or “pm”. For instance, to specify 3pm and 11am, the arguments would be time1=3, suffix1=”pm”, time2=11, suffix2=”am”. Assume that all times are in whole numbers, i.e., there is no time such as 11:20.