Reversing a string

May 18, 2020

1 Reversing a String

The goal in this notebook will be to get practice with a problem that is frequently solved by recursion: Reversing a string.

Note that Python has a built-in function that you could use for this, but the goal here is to avoid that and understand how it can be done using recursion instead.

1.0.1 Exercise - Write the function definition here

In [ ]:
```
# Code

def reverse_string(input):
    """
    Return reversed input string

    Examples:
       reverse_string("abc") returns "cba"

    Args:
      input(str): string to be reversed

    Returns:
      a string that is the reverse of input
    """

    # TODO: Write your recursive string reverser solution here

    pass
```

1.0.2 Test - Let's test your function

In [ ]:
```
# Test Cases

print ("Pass" if  ("" == reverse_string("")) else "Fail")
print ("Pass" if  ("cba" == reverse_string("abc")) else "Fail")
```

In [ ]:
```
# Solution
"""
```

```
RECURSIVE FUNCTION
Args: input(str): string to be reversed
Returns: a string that us reversed of input
"""
def reverse_string(input):

    # (Recursion) Termination condition / Base condition
    if len(input) == 0:
        return ""

    else:
        first_char = input[0]

        '''
        The `slice()` function can accept upto the following three arguments.
        - start: [OPTIONAL] starting index. Default value is 0.
        - stop: ending index (exclusive)
        - step_size: [OPTIONAL] the increment size. Default value is 1.

        The return type of `slice()` function is an object of class 'slice'.
        '''
        the_rest = slice(1, None)     # `the_rest` is an object of type 'slice' class
        sub_string = input[the_rest]  # convert the `slice` object into a list

        # Recursive call
        reversed_substring = reverse_string(sub_string)

        return reversed_substring + first_char
#-------------------------------------------------#
'''
**Time and Space Complexity Analysis**
Each recursive call to the `reverse_string()` function will create
a new set of local variables - first_char, the_rest, sub_string, and reversed_substring.
Therefore, the space complexity of a recursive function would always be proportional to
maximum depth of recursion stack.
The time complexity for this function will be  O(k*n), where k is a constant and n is th
number of characters in the string (depth of recursion stack).
'''
```

Show Solution

In [ ]: