module3

January 23, 2024

0.1 Module 3

In this assignment, you will implement some functions related to strings, lists, sets and tuples. Each function has been defined for you, but without the code. See the docstring in each function for instructions on what the function is supposed to do and how to write the code. It should be clear enough. In some cases, we have provided hints to help you get started.

[1]:
```
############################################################
### EXECUTE THIS CELL BEFORE YOU TO TEST YOUR SOLUTIONS ###
############################################################

import unittest
from nose.tools import assert_equal, assert_true
import nose.tools
```

[2]:
```
def concatenate(strings):
    """
    Concatenates the given list of strings into a single string.
    Returns the single string.
    If the given list is empty, returns an empty string.

    For example:
    - If we call concatenate(["a","b","c"]), we'll get "abc" in return
    - If we call concatenate([]), we'll get "" in return

    Hint(s):
    - Remember, you can create a single string from a list of multiple strings␣
 ↪by using the join() function
    """
    # your code here
    x = "".join(strings)
    return x
```

[3]:
```
############################
### TEST YOUR SOLUTION ###
############################

lst = ["a","b","c"]
```

```
assert_equal("abc",concatenate(lst))
lst = []
assert_equal("",concatenate(lst))
print("Success!")
```

```
Success!
```

[4]:
```
def all_but_last(seq):
    """
    Returns a new list containing all but the last element in the given list.
    If the list is empty, returns None.

    For example:
    - If we call all_but_last([1,2,3,4,5]), we'll get [1,2,3,4] in return
    - If we call all_but_last(["a","d",1,3,4,None]), we'll get ["a","d",1,3,4]␣
 ↪in return
    - If we call all_but_last([]), we'll get None in return
    """
    # your code here
    if len(seq) == 0 :
        return None
    else:
        seq.pop()
        return seq
```

[5]:
```
############################
### TEST YOUR SOLUTION ###
############################

lst = []
assert_equal(None,all_but_last(lst))
lst = [1,2,3,4,5]
nose.tools.assert_list_equal([1,2,3,4],all_but_last(lst))
lst = ["a","d",1,3,4,None]
nose.tools.assert_list_equal(["a","d",1,3,4],all_but_last(lst))
print("Success!")
```

```
Success!
```

[6]:
```
def remove_duplicates(lst):
    """
    Returns the given list without duplicates.
    The order of the returned list doesn't matter.

    For example:
    - If we call remove_duplicates([1,2,1,3,4]), we'll get [1,2,3,4] in return
    - If we call remove_duplicates([]), we'll get [] in return
```

```
    Hint(s):
    - Remember, you can create a set from a string, which will remove the␣
↪duplicate elements
    """

    # your code here
    lst = [*set(lst)]
    return lst
```

[7]:
```
############################
### TEST YOUR SOLUTION ###
############################

lst = [1,3,4,3,4,5,2]
nose.tools.assert_count_equal([1,3,4,5,2],remove_duplicates(lst))
lst = []
nose.tools.assert_count_equal([],remove_duplicates(lst))
print("Success!")
```

```
Success!
```

[8]:
```
def reverse_word(word):
    """
    Reverses the order of the characters in the given word.

    For example:
    - If we call reverse_word("abcde"), we'll get "edcba" in return
    - If we call reverse_word("a b c d e"), we'll get "e d c b a" in return
    - If we call reverse_word("a  b"), we'll get "b  a" in return
    - If we call reverse_word(""), we'll get "" in return

    Hint(s):
    - You can iterate over a word in reverse and access each character
    """

    # your code here
    l=[]
    for i in [*word]:
        l.insert(0,i)
    return "".join(l)
```

[9]:
```
############################
### TEST YOUR SOLUTION ###
############################

word = "abcdefg"
```

```
assert_equal("gfedcba",reverse_word(word))

word = "a b c d e f g"
assert_equal("g f e d c b a",reverse_word(word))

word = "a b"
assert_equal("b a",reverse_word(word))

word = ""
assert_equal("",reverse_word(word))
print("Success!")
```

```
Success!
```

[10]:
```
def divisors(n):
    """
    Returns a list with all divisors of the given number n.
    As a reminder, a divisor is a number that evenly divides another number.
    The returned list should include 1 and the given number n itself.
    The order of the returned list doesn't matter.

    For example:
    - If we call divisors(10), we'll get [1,2,5,10] in return
    - If we call divisors(1), we'll get [1] in return
    """
    # your code here
    i = 1
    l = []
    while i <= n :
        if (n % i==0) :
            l.append(i)
        i = i + 1
    return l
```

[11]:
```
############################
### TEST YOUR SOLUTION ###
############################

number = 10
nose.tools.assert_count_equal([1,2,5,10],divisors(number))

number = 1
nose.tools.assert_count_equal([1],divisors(number))

number = 7
nose.tools.assert_count_equal([1,7],divisors(number))
print("Success!")
```

Success!

[12]:
```
def capitalize_or_join_words(sentence):
    """
    If the given sentence starts with *, capitalizes the first and last letters␣
↪of each word in the sentence,
    and returns the sentence without *.
    Else, joins all the words in the given sentence, separating them with a␣
↪comma, and returns the result.

    For example:
    - If we call capitalize_or_join_words("*i love python"), we'll get "I LovE␣
↪PythoN" in return.
    - If we call capitalize_or_join_words("i love python"), we'll get␣
↪"i,love,python" in return.
    - If we call capitalize_or_join_words("i love    python  "), we'll get␣
↪"i,love,python" in return.

    Hint(s):
    - The startswith() function checks whether a string starts with a␣
↪particualr character
    - The capitalize() function capitalizes the first letter of a string
    - The upper() function converts all lowercase characters in a string to␣
↪uppercase
    - The join() function creates a single string from a list of multiple␣
↪strings
    """
    # your code here
    if sentence.startswith('*'):
        sentence = sentence[1:]
        words = sentence.split()
        processed_words = [word[0].upper() + word[1:-1] + word[-1].upper() for␣
↪word in words]
        processed_sentence = ' '.join(processed_words)
        return processed_sentence[1:]
    else:
        words = sentence.split()
        joined_sentence = ','.join(words)
        return joined_sentence
```

[13]:
```
############################
### TEST YOUR SOLUTION ###
############################

string = "*i love python"
assert_equal("I LovE PythoN",capitalize_or_join_words(string))
```

```
string = "i love python"
assert_equal("i,love,python",capitalize_or_join_words(string))

string = "i love    python  "
assert_equal("i,love,python",capitalize_or_join_words(string))
print("Success!")
```

```
Success!
```

[14]:
```
def move_zero(lst):
    """
    Given a list of integers, moves all non-zero numbers to the beginning of␣
 ↪the list and
    moves all zeros to the end of the list.  This function returns nothing and␣
 ↪changes the given list itself.

    For example:
    - After calling move_zero([0,1,0,2,0,3,0,4]), the given list should be␣
 ↪[1,2,3,4,0,0,0,0] and the function returns nothing
    - After calling move_zero([0,1,2,0,1]), the given list should be␣
 ↪[1,2,1,0,0] and the function returns nothing
    - After calling move_zero([1,2,3,4,5,6,7,8]), the given list should be␣
 ↪[1,2,3,4,5,6,7,8] and the function returns nothing
    - After calling move_zero([]), the given list should be [] and the function␣
 ↪returns nothing
    """
    # your code here
    count = lst.count(0)
    non_zero_nums = [num for num in lst if num != 0]
    lst.clear()
    lst.extend(non_zero_nums + [0] * count)
```

[15]:
```
############################
### TEST YOUR SOLUTION ###
############################

lst = [0,1,0,2,0,3,0,4]
assert_equal(None,move_zero(lst))
nose.tools.assert_list_equal([1,2,3,4,0,0,0,0],lst)

lst = []
move_zero(lst)
nose.tools.assert_list_equal([],lst)

lst = [0,0,0,0,0,0,0,0,0]
move_zero(lst)
nose.tools.assert_list_equal([0,0,0,0,0,0,0,0,0],lst)
```

```
lst = [1,2,3,4,5,6,7,8]
move_zero(lst)
nose.tools.assert_list_equal([1,2,3,4,5,6,7,8],lst)
print("Success!")
```

```
Success!
```

[16]:
```
def main():
    """
    Calls all the functions above to see whether they've been implemented␣
 ↪correctly.
    """

    # test concatenate
    print("test concatenate")
    word = concatenate(["b", "e", "a", "t", "l", "e", "s"])
    print(word == "beatles")
    print("=" * 50)

    # test all_but_last
    print("test all_but_last")
    seq = all_but_last(["john", "paul", "george", "ringo", "tommy"])
    print(seq == ["john", "paul", "george", "ringo"])
    print("=" * 50)

    # test remove_duplicates
    print("test remove_duplicates")
    res = remove_duplicates([1, 3, 4, 2, 1])
    print(res == [1, 3, 4, 2])
    print("=" * 50)

    # test reverse_word
    print("test reverse_word")
    res = reverse_word("alphabet")
    print(res == "tebahpla")
    print("=" * 50)

    # test divisors
    print("test divisors")
    res = divisors(120)
    print(set(res) == set([1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60,␣
 ↪120]))
    print("=" * 50)

    # test capitalize_or_join_words
    print("test capitalize_or_join_words")
```

```
    print("Result for String Start With *: ")
    # Should return "I LovE CodinG AnD I'M HavinG FuN"
    res = capitalize_or_join_words("*i love coding and i'm having fun")
    print(res == "I LovE CodinG AnD I'M HavinG FuN")

    print("Result for Other String: ")
    # Should print "I,love,coding,and,I'm,having,fun"
    res = capitalize_or_join_words("I love coding and I'm having fun")
    print(res == "I,love,coding,and,I'm,having,fun")
    print("=" * 50)

    # test move_zero
    print("test move_zero")
    lst = [0, 1, 0, 2, 0, 3, 4, 0]
    print("Before move,the list looks like\n", lst)
    move_zero(lst)
    print("After move,the list looks like\n", lst)
    print("=" * 50)

#This will automatically run the main function in your program
#Don't change this
if __name__ == '__main__':
    main()
```

```
test concatenate
True
==================================================
test all_but_last
True
==================================================
test remove_duplicates
False
==================================================
test reverse_word
True
==================================================
test divisors
True
==================================================
test capitalize_or_join_words
Result for String Start With *:
True
Result for Other String:
True
==================================================
test move_zero
Before move,the list looks like
```

```
 [0, 1, 0, 2, 0, 3, 4, 0]
After move,the list looks like
 [1, 2, 3, 4, 0, 0, 0, 0]
================================================
```