CS5001 Spring 2022 EXAM 2

Question 7 (5 points)

Write a predicate function that meets the following specifications:

- **Name:** `is_cs5001`
- **Parameter: a string**.
- **Returns:** A Boolean indicating if the string passed in contains the string `"CS5001"`.
- **Examples:**
    - `is_cs5001("I love CS5001 5001 times")` returns `True`
    - `is_cs5001("I love 5001 CS students")` returns `False`

Question 8 (10 points):

Write a function that adds a key/value pair to a dictionary, but only if the key is not there already. Your function should meet the following specifications:

- **Name: append**
- **Parameters:** a dictionary, a key (any legal data type), and a value (any data type)
- **Returns:** an updated version of the dictionary, with the given key/value pair added to it.
  If the key/value pair already exists in the given dictionary, **return the dictionary unmodified.**
- **Example:**
  assume d is an already created dictionary. The following updates the dictionary with the key/value pair of "a string" as the key and "another string" as the value
  `d = append(d, "a string", "another string")`

Question 9 (10 points):

Write a **recursive** function that meets the following specifications. To earn full credit, your implementation MUST be recursive (not iterative, and not a simple reverse slice):

- **Name**: **recursive_reverse**
- **Parameter**: a string
- **Returns**: the original string passed in, reversed
- **Examples**:
  `recursive_reverse("Hello")` returns `"olleH"` from this function.
  `recursive_reverse("")` returns `""` from this function.

Question 10 (15 points)

Write a function that takes as input a list that contains nested lists of two (2) elements each. For each nested list (call them "pairs"), swap the pairs' values while maintaining the pairs position in the enclosing list. This function **should NOT modify (mutate)** the original list passed in.

Specification:

- **Name: reverse_list_of_pairs**
- **Parameters:** a list that contains nested lists. Each "inner" list holds two values.
- **Returns:** a list of lists. Each "inner" list holds two values
- **Example:** Given a call to: `reverse_list_of_pairs([[1,2], [3,4], [5,6]])`
  returns the list-of-lists: `[[2, 1], [4, 3], [6, 5]]`

Question 11 (10 points)

Write a function that takes as input a list that contains a list of integers. Return a list that contains the even numbers that were in the original list. This function should NOT modify (mutate) the original list passed in.

Specification:

- **Name: evens**
- **Parameters:** a list of integers.
- **Returns:** a list of integers. The list holds the even numbers from the list passed in
- **Example:** Given a call to: `evens([1,2,3,4])`
  returns the list: `[2,4]`

Question 12 (15 points)

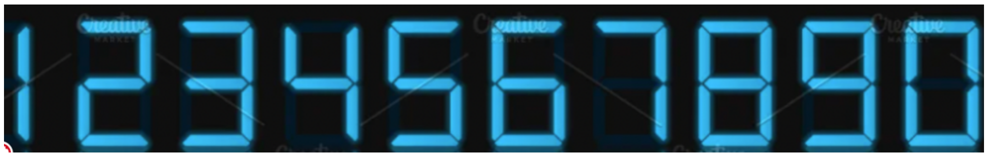
Palindromic numeric ambigrams

A **palindrome** is a word, phrase or number that reads the same forwards and backwards. An **ambigram** something (such as an image of a written word or phrase) that is intended or able to be oriented in either of two ways for viewing. When you flip an ambigram upside down, it can create the same image OR another word.

For example, with a specialized font, the word "ambigram" is an ambigram when you rotate it 180 degrees.

As another example, the word "mom" becomes "wow" when rotated.

For this question, we're keeping it simple and going to limit ourselves to integers only. We will use the following "digital clock" number font for illustration



Given this, the numbers **1, 2, 5, 8 and 0 are exactly the same** if they are rotated 180 degrees in either direction.

Write a function that meets the following specifications:

- **Name**: **is_ambigram**
- **Parameter**: a non-negative integer n
- **Returns**: **True** if the number is both a **palindrome** AND an **ambigram**, **False** otherwise
- **Example**: The date Dec 02, 2021 (in USA short format: 12/02/2021) is a palindromic ambigram. Calling `is_ambigram(12022021)` returns `True`

Question 13 (15 points)

Electric Cars

Given the `TestElectricCar` class below, write a class called ElectricCar (exact same spelling!) that will pass the tests contained in our test class. **Do NOT change anything in our test class**!

Your `ElectricCar` class must provide the following methods to adequately pass our tests:

```
__init__(self, name, battery)
""" Initializes car with name and battery size. Minimum
battery value is 50.0; Max battery value is 200.0. If
battery value passed in is less than minimum or greater
than maximum, raise a ValueError"""

calculate_range(self, state_of_charge)
""" Returns a numeric value of the current range. State of
charge is a value between 0.0 and 1.0 (represents percent
charge remaining). If state_of_charge is less than 0.0 or
more than 1.0, raise a ValueError.
Range is calculated by this formula:
battery * state_of_charge * winter_factor * 3
where:
battery is the battery size initialized in your constructor
state_of_charge is the charge value passed in to this
method
winter_factor == 1 if NOT winter, or 0.7 if winter is True
3 is a constant representing average battering operating
time
"""

set_winter(self, value)
""" Value True or False if car is operating in winter"""

get_winter(self, value)
""" Returns True or False based on if car is operating in
winter or not"""

get_name(self)
""" Returns name of car"""
```

```
import unittest

class TestElectricCar(unittest.TestCase):
    '''
        __init__, set_winter, calculate_range
    '''

    def test_init(self):
        tesla = ElectricCar("Tesla Y", 78)
        self.assertEqual(tesla.get_name(), "Tesla Y")
        self.assertFalse(tesla.get_winter())

        machE = ElectricCar("Ford Mach-E", 80)
        self.assertEqual(machE.get_name(), "Ford Mach-E")
        self.assertFalse(machE.get_winter())

    def test_set_winter(self):
        tesla = ElectricCar("Tesla Y", 78)
        self.assertFalse(tesla.get_winter())
        tesla.set_winter(True)
        self.assertTrue(tesla.get_winter())

        tesla.set_winter(False)
        self.assertFalse(tesla.get_winter())

    def test_bad_init_low(self):
        with self.assertRaises(ValueError):
            car = ElectricCar("eYugo", 49.9)

    def test_bad_init_high(self):
        with self.assertRaises(ValueError):
            car = ElectricCar("eYugo", 200.1)


    def test_bad_range_low(self):
        with self.assertRaises(ValueError):
            car = ElectricCar("eYugo", 80)
            car.calculate_range(-1)

    def test_bad_range_high(self):
        with self.assertRaises(ValueError):
            car = ElectricCar("eYugo", 80)
            car.calculate_range(1.2)

    def test_range(self):
        tesla = ElectricCar("Tesla Y", 78)
        machE = ElectricCar("Ford Mach-E", 80)

        self.assertEqual(tesla.calculate_range(1), 234)
        self.assertEqual(tesla.calculate_range(0.5), 117)

        machE.set_winter(True)
        self.assertAlmostEqual(machE.calculate_range(0.9),
151.2)
        machE.set_winter(False)
        self.assertEqual(machE.calculate_range(0.9), 216)


def main():
    unittest.main(verbosity = 3)
main()
```