Assignment 6

Use the following naming scheme for your program files: *aassignment#pproblem#yourname*`.py`. So your program for problem 1 on this assignment will be named `a6p1bob.py` and your program for problem 2 will be named `a6p2bob.py` (adjust to be your name). Please submit all of your .py files to the Moodle dropbox.

Problems

1. Insert the necessary function definitions into the code below so that it will work properly (i.e. as shown in the sample runs below it).

   Notes:

   - 0 is a one-digit, even, unsigned number.
   - You may not make any changes to the code below, i.e. your functions have to fit the code, you are not allowed to modify the code to fit your functions.

```
# a6p1v1.py
# Just some practice writing little functions.
# Insert the necessary functions here.

line(60, '=')
print('Function practice')
line(60, '-')
num = int(input('Give me an integer value: '))
print('Your number contains', ndigits(num), end = ' ')
print('digits, is', even_or_odd(num), end = '')
if ispos(num):
    print(', and is positive.')
else:
    print(', but is not positive.')
line(60, '-')
```

```
============================================================
Function practice
------------------------------------------------------------
Give me an integer value: 78931
Your number contains 5 digits, is odd, and is positive.
------------------------------------------------------------
```

```
============================================================
Function practice
------------------------------------------------------------
Give me an integer value: -122
Your number contains 3 digits, is even, but is not positive.
------------------------------------------------------------
```

Hint: There are several ways of tackling the `ndigits` function.

- You can find the number of digits by counting how many times you can divide a number by 10 before you get 0, e.g. 7 can be divided by 10 once, while 732 can be divided 3 times.
- You can use the logarithm of the number to the base 10.
- You can convert the number to a string and then use **len()** to see how long the string is.

The next three problems all involve writing functions to determine if a list of card numbers represents a straight. A straight is a hand in which the cards' face values form a sequence, e.g. 3, 4, 5, 6, 7 and 6, 4, 7, 3, 5 are both straights (even though the numbers in the second case are not in order), but 3, 4, 5, 7, 8 and 4, 5, 6, 5, 7 are not (the first, because 6 is missing, and the second because of the duplicate 5s). Note that the cards do not have to be in the same suit, e.g. the hand containing 2♣, 3♦, 4♣, 5♠ and 6♥ is a straight.

There are numerous computational approaches that can be taken to decide if a list of card numbers is a straight or not. Each of the following problems describes one of them and provides pseudocode for it. **Your job is to translate each pseudocode description into a Python function**.

(Hint: You will notice that all three approaches contain the operation "replace the card numbers in the copy with numbers representing their face values". To avoid rewriting, that would make a good function, no?)

2. One approach to seeing if the cards in the list are a straight is to sort them in ascending order and then see if each one is one larger than the previous one. That is we take 6, 4, 7, 3, 5 and sort it to get 3, 4, 5 ,6, 7, and then we make sure the second entry (4) is one larger than the first (3), and that the third entry (5) is one larger than the second

(4), and so on. If they all are it's a straight, but if we find neighbouring values that are the same, or that are more than 1 apart it's not a straight.

Pseudocode:

```
def isStraight(hand):
    '''Return True if the cards in hand are a straight, and
False otherwise.'''
    make a copy of hand (so we don't change it) and work with
the copy from now on
    replace the card numbers in the copy with numbers
representing their face values
        (e.g. 1 for Ace, 2 for Two, ... and 13 for King)
    sort the copy in ascending numerical order
    Check that each value is one larger than the previous value:
    assume the cards are a straight by setting a flag to True
    for each value in the copy starting with the second value
    and going to the last value
        if this value is not one greater than the previous value
            set the flag to False
    if the flag is still True
        it's a straight
    otherwise
        it's not
```

3. Our second approach begins the same way as the first, i.e. by replacing card numbers by face values and sorting them, but then we compare the sorted list to a test list with the same starting value. What test list? Suppose I tell you I have a five card straight starting from a face value of 3. You could say "Oh you have a 3, a 4, a 5, a 6 and a 7." The idea here is to build the necessary test list and compare it to the actual list and to see if they match. If they do it's a straight, otherwise it's not.

Pseudocode:

```
def isStraight(hand):
    '''Return True if the cards in hand are a straight, and
    False otherwise.'''
    make a copy of hand (so we don't change it) and work with
    the copy from now on
    replace the card numbers in the copy with numbers
    representing their face values
     (e.g. 1 for Ace, 2 for Two, ... and 13 for King)
    sort the copy in ascending numerical order
    find the minimum face value in the list
    build a new list containing the values a straight starting
     with that
        minimum value would contain, i.e. a list of consecutive
         numbers starting with the min value and having the same
         number of values in it as the hand, e.g. min value = 3
         and hand size = 5 implies a test list of [3, 4, 5, 6,
         7]
         (Hint: Use the range command to build the test list.)
    if the test list equals the sorted copy
        it's a straight
    else
        it's not
```

4. Our final approach is based on the observation that the cards in a straight must all be different, and that the first and last cards' face values will differ by the length of the hand minus 1, e.g. in a five card hand the face value of the last card will be 5 minus 1 or 4 greater than the first card (as it is for example in [3, 4, 5, 6, 7] where 7 is 4 greater than 3). Of course the hand [3, 3, 5, 7, 7] also has the right range of values but isn't a straight because the face values in it are not unique (see the first condition in the first sentence above "the cards in a straight must all be different"). Which leads to the second test in the pseudocode below:

```
def isStraight(hand):
    '''Return True if the cards in hand are a straight, and
    False otherwise.'''
    make a copy of hand (so we don't change it) and work with
     the copy from now on
    replace the card numbers in the copy with numbers
     representing their face values
    find the minimum face value (Hint: Use the min function.)
    find the maximum face value (Hint: Can't you guess?)
    # If the largest and smallest values are the right distance
     apart
    # e.g. in a five card hand they 4 apart
    if max - min equals length of the list - 1 then
        # and all the values in the list are different
        # i.e. they all occur exactly once
        set the counter of unique values to 0
        for each value in the list
            if the number of times it occurs in the list is 1
                (Hint: Use the count method.)
                add one to the counter of unique values
        if the count of unique values equals the length of the
         list
            it's a straight
        otherwise
            it's not
     else
        it's not a straight
```

You can use the test program below to evaluate your code -- I will in marking it, though I may add to the set of test cases given below — or you can write and use your own test program.

```
# isstraight_test.py
# This program tests the function isStraight on several sample
hands
# of cards.
from a6p2name import isStraight
#from a6p3name import isStraight
#from a6p4name import isStraight

# TEST_HANDS is a list containing the hands of cards to use in
testing
# the function isStraight, and the correct result for each hand.
# Note that not all hands have five cards, and some are straights,
# while some are not.
TEST_HANDS = [
    [[ 1, 2, 3, 4, 5 ], True],
    [[ 5, 4, 3, 2, 1 ], True],
    [[ 14, 0, 28, 42, 4 ], True],
    [[ 1, 2, 3, 4, 5, 6 ], True],
    [[ 5 ], True],
    [[ 1, 2, 2, 4, 5 ], False],
    [[ 1, 2, 2, 5, 5 ], False],
    [[ 1, 3, 5, 7, 9 ], False]
]

print('Testing isStraight ... ')
# Loop through the list of TEST_HANDS, to test each sample hand.
for test in TEST_HANDS:
    # If the function isStraight does not return the correct
    # result...
    if isStraight(test[0]) != test[1]:
        # ... display an error message
        print('isStraight fails on', test[0] )
print('Done.\n')
```