Homework 3

Problem 1: Tuple Input

```
# Problem 1 Input Tuple
'''Reads strings from user or a file, then uses a tuple to typecast each item
    Parameters:
        prompt: string for input prompt
        types: a tuple of types that match user input
        sep: separator used in parsing the user input. Defaule = ","
'''
from testif import testif

# function will pass these variables containing expected values
prompt_str = "Enter first name, last name, Age, ID, Fulltime(y/n or leave blank)\
              separate items with ','\n   > "
types_tup = (str, str, float, int, bool)
car_types_tup = (str, str, float, int)

def cast(typ, item):
    """Returns the typed value of the item string"""
    if typ == bool:
        if item.lower() != 'y' or item.strip() == "":
            return False
        else:
            return True
    return typ(item)

def input_tuple(prompt, types, sep=','):
    """Success: Returns converted values. Failure: Returns empty tuple"""

    word_list = input(prompt).split(sep)
    if len(word_list) != len(types):
        print("Error: Number of items entered is not equal to number of types to
parse")
        return ()

    try:
        typed_list = []
        for i, word in enumerate(word_list):
            typed_list.append(cast(types[i], word.strip()))
        return tuple(typed_list)

    except ValueError:
```

```
        print("Error: Parsing strings failed")
        return ()

def input_tuple_lc(prompt, types, sep=","):
    """Same as input_tuple but using list comprehension"""
    word_list = input(prompt).split(sep)
    if len(word_list) != len(types):
        print("Error: Number of items entered is not equal to number of types to
parse")
        return ()

    try:
        return tuple([cast(types[i], word.strip()) for i, word in
enumerate(word_list)])

    except ValueError:
        print("Error: Parsing strings failed")
        return ()

def convert(string, types, sep=","):
    """Works like input_tuple_lc without the input function"""
    word_list = string.split(sep)
    if len(word_list) != len(types):
        print("Error: Number of items entered is not equal to number of types to
parse")
        return ()

    try:
        return tuple([cast(types[i], word.strip()) for i, word in
enumerate(word_list)])

    except ValueError:
        print("Error: Parsing strings failed")
        return ()

def read_tuple(file_obj, types, sep=","):
    """Same as input_tuple but reades input from a file."""
    line = file_obj.readline()
    return convert(line, types, sep)

def read_list_of_tuples(f_obj, typs, sep=','):
    """Returns a list of tuples rather than just one"""
    car_vals = []
    for line in f_obj:
        if len(line.split(sep)) != len(typs):
```

```
            return ()
        car_vals.append(tuple([typ(el) for typ, el in zip(typs,
line.rstrip('\n').split(sep))]))
    return car_vals

def test_input_tuples():
    '''Testing functions that don't read from file'''
    student_tuple = input_tuple(prompt_str, types_tup, ',')
    testif(student_tuple == ('Dan', 'Pettus', 27.0, 2332, True), "input_tuple"\
            , "input_tuple passed", "input_tuple failed")

    student_tuple_lc = input_tuple_lc(prompt_str, types_tup, ',')
    testif(student_tuple_lc == ('Dan', 'Pettus', 27.0, 2332, False),
"input_tuple_lc"\
            , "input_tuple_lc passed", "input_tuple_lc failed")

    # Test for failures
    testif(student_tuple == ('Dan', 'Pettus', 27.0, 2332, True), "input_tuple"\
            , "input_tuple passed", "input_tuple failed")
    testif(student_tuple_lc == ('Dan', 'Pettus', 27.0, 2332, False),
"input_tuple_lc"\
            , "input_tuple_lc passed", "input_tuple_lc failed")

def test_read_files():
    '''Test: returns 1 tuple per call'''
    fin = open("cars.csv", "r")
    car1 = read_tuple(fin, car_types_tup, ",")
    car2 = read_tuple(fin, car_types_tup, ",")
    fin.close()
    testif(car1 == ('Lada', 'Niva', 19.5, 1987) and car2 == ('Porsche', '911
Turbo S', 17.5, 1989)\
        , 'read_tuple', 'read_tuple passed', 'read_tuple failed')

    fin = open("cars.csv", "r")
    car_list = read_list_of_tuples(fin, car_types_tup, ',')
    testif(car_list == [('Lada', 'Niva', 19.5, 1987), ('Porsche', '911 Turbo S',
17.5, 1989)]\
        , 'read_list_of_tuples', 'read_list_of_tuples passed',
'read_list_of_tuples failed')
    fin.close()

    # Failed Tests
    fin = open("cars_fail.csv", "r")
    car1 = read_tuple(fin, car_types_tup, ",")
    car2 = read_tuple(fin, car_types_tup, ",")
```

```
    fin.close()
    testif(car1 == ('Lada', 'Niva', 19.5, 1987) and car2 == ('Porsche', '911
Turbo S', 17.5, 1989)\
        , 'read_tuple', 'read_tuple passed', 'read_tuple failed')

    fin = open("cars_fail.csv", "r")
    car_list = read_list_of_tuples(fin, car_types_tup, ',')
    testif(car_list == [('Lada', 'Niva', 19.5, 1987), ('Porsche', '911 Turbo S',
17.5, 1989)]\
        , 'read_list_of_tuples', 'read_list_of_tuples passed',
'read_list_of_tuples failed')
    fin.close()

def test_read_list_of_tuples():
    '''Test: returns list of tuples'''
    f_obj = open("cars.csv", "r")
    print(read_list_of_tuples(f_obj, car_types_tup, ','))
    f_obj.close()

def main():
    """Main Function"""
    test_input_tuples()
    test_read_files()

main()
```

Problem 2: Pythagorean Triples

```
# Pythagorean Numbers Revisited
"""This program takes one positive int argument n and returns a list with
Pythagorean Triples"""

import math

def compute_Pythagoreans(n):
    '''Returns List of Tuples'''
    if n > 0 and type(n) == int:
        return [(a, b, c) for a in range(3, n+1) for b in range(4, n+1) for c in
range(5,n+1) if a**2+b**2==c**2]
    else:
        return []

n = int(input("Enter a number greater than zero: "))
print(compute_Pythagoreans(n))
```

Problem 3: Lebron Worship

```
'''Program returns and graphs CSV data'''
import pylab

def parse_token(token, str_pos_lst, token_pos):
    '''Returns str for tokens in str_pos_list; Returns floats for all others'''
    if token_pos in str_pos_lst:
        return str(token)
    else: return float(token)

def parse_line(line, string_pos_lst, sep):
    '''Parsed line into tokens and converst them to str() or float()'''
    return [parse_token(token, string_pos_lst, token_pos)
            for token_pos, token in enumerate(line.rstrip('\n').split(sep))]

def get_csv_data(f_obj, string_pos_lst, sep=","):
    '''Returns a nested list called data_lst'''
    if sep != ',':
        print("invalid separator: file is comma separated")
        return []

    data_lst = []
    header_row = f_obj.readline()
    data_lst.append(header_row.rstrip('\n').split(sep))
    for data_row in f_obj:
        try:
            data_lst.append(parse_line(data_row, string_pos_lst, sep))
        except ValueError:
            continue
    return data_lst

def print_data(data):
    '''Prints data in lines'''
    for line in data:
        print(line)

def get_index(row, heading):
    '''Returns index of the desired column within the given row'''
    return row.index(heading)

def fill_col(data_lst, col):
    return [row[col] for row in data_lst]

def get_columns(data_lst, cols_lst):
```

```
    '''Returns column data for each heading given in columns list'''
    if cols_lst == []:
        return []
    col_positions = []
    for col in cols_lst:
        if col in data_lst[0]:
            col_positions.append(get_index(data_lst[0], col))
        else:
            continue
    return [fill_col(data_lst, col) for col in col_positions]

    # col_pos_lst = [get_index(data_lst[0], col_name) for col_name in cols_lst]
    # return [fill_col(data_lst, col) for col in col_pos_lst]


def main():
    '''Main Function'''
    bb_file = open("lb-james.csv", "r")
    bb_lst = get_csv_data(bb_file, [0, 2, 3, 4], ",")
    columns = get_columns(bb_lst, ['3P%', '2P%', 'FT%'])

    x = [i for i in range(4, len(columns[0])+3)]
    y_3fg = [columns[0][i] for i in range(1, len(columns[0]))]
    y_2fg = [columns[1][i] for i in range(1, len(columns[1]))]
    y_ft = [columns[2][i] for i in range(1, len(columns[2]))]
    xy1 = zip(x, y_2fg)
    xy2 = zip(x, y_3fg)
    xy3 = zip(x, y_ft)
    x1 = []
    y1 = []
    x2 = []
    y2 = []
    x3 = []
    y3 = []

    for x, y in xy1:
        x1.append(x)
        y1.append(y)
    for x, y in xy2:
        x2.append(x)
        y2.append(y)
    for x, y in xy3:
        x3.append(x)
        y3.append(y)
```

```
    pylab.title('Lebrons FG percentage')
    fg2, = pylab.plot(x1, y1, color = 'blue', label = '2FG')
    fg3, = pylab.plot(x2, y2, color = 'red', label = '3FG')
    fth, = pylab.plot(x3, y3, color = 'green', label = 'FT')
    pylab.legend(handles=[fg2, fg3, fth])
    pylab.xlim(1, 15)
    pylab.ylim(0, 1)
    pylab.show()

main()
```