Problem 1

```
#Michael Quach

#Problem 1. Tuple Input

#a) Write a function called input_tuple (in a new file p1.py)
#that reads from the terminal a sequence of objects with types
#provided by a tuple given as parameter and that returns the
#sequence of objects read as a tuple. The function will iterate
#through the types tuple and will try to parse the input string
#based on the types.
#The function takes the following parameters:
#    -- prompt: a string to be displayed as input prompt
#    -- types: a tuple of type objects that must match the types
#    of the data entered by the user and returned by the function.
#    The only types supported are int, float, str, bool
#    -- sep: a string that separates the input objects, with
#    default value ",".
def input_tuple(prompt, types, sep):
    try:
        data = input(prompt)
        data = data.split(sep)

        for i in range(len(types)):
            data[i] = tuple(types[i](data[i]))
        return data

    except ValueError:
        print("Error while parsing input. Please double-check your input. Terminating...")
        return ()
    except:
        print("Oops! Error encountered. Terminating program.")
        return ()
##Take input, get string of stuff
##Divide stuffs using "sep"arator and .split
##Convert stuffs into appropriate types based on "types" list

#The function returns one of:
#    -- if parsing the data from the user according to the types
#    tuple succeeds, then it returns the tuple with the converted
#    data to Python objects,
#    -- if parsing the data from the user according to the types
#    tuple fails, then it returns the empty tuple ( ).
```

```
#
#Error handling: the function must not raise/throw errors related
#to parsing or input/read operations. Use proper error handling
#with try/except. In case of any error, print an error message to
#the terminal and return the empty tuple ().
#
#b) Write a function called input_tuple_lc that is identical to input_tuple
#except that it uses list comprehension(s).
#
def input_tuple_lc(prompt, types, sep):
    try:
        data = input(prompt)
        data = data.split(sep)

        data = tuple([types[i](data[i]) for i in range(len(types))])
        return data

    except ValueError:
        print("Error while parsing input. Please double-check your input. Terminating...")
        return ()
    except:
        print("Oops! Error encountered. Terminating program.")
        return ()
#c) Write a function read_tuple that works similarly to input_tuple,
#but instead of reading input from the terminal, it reads text from a
#file object passed as argument. If this function uses correctly a list
#comprehension you get 2 extra points.
#
#The function  read_tuple takes the following parameters:
#    -- file_obj: an object representing an open text file;
#    e.g. opened with open("filename","r").
#    -- types: a tuple of type objects that must match the types of
#    the data entered by the user and returned by the function.
#    The only types supported are int, float, str, bool
#    -- sep: a string that separates the input objects, with default value ",".
#The function returns one of:
#    -- if parsing the data from the user according to the types tuple
#    succeeds, then it returns the tuple with the converted data
#    -- if parsing the data from the user according to the types tuple
#    fails, then it returns the empty tuple ( ).
#
def read_tuple(file_obj, types, sep):
    try:
```

```
        data = file_obj.read()
        data = data.split(sep)

        data = tuple([types[i](data[i]) for i in range(len(types))])
        return data

    except ValueError:
        print("Error while parsing input. Please double-check your input. Terminating...")
        return ()
    except:
        print("Oops! Error encountered. Terminating program.")
        return ()
    finally:
        file_obj.close()
#Error handling: the function must not raise/throw errors related
#to parsing or input/read operations. Use proper error handling with
#try/except. In case of any error, print an error message to the
#terminal and return the empty tuple ().
#
#c) In the main program (p1.py) write code that tests both functions.
#Using the testif function from the Unit 2 module for writing your
#tests gives you  2 extra credit points.
#Function used for testing.
#param b: boolean, normally a tested condition: true if test passed, false otherwise
#param msgOK: string to be printed if param b==True (test condition true)
#param msgFailed: string to be printed if param b==False
#returns b
def testif(b, testname, msgOK="", msgFailed=""):
    if b:
        print("Success:", testname, ";", msgOK)
    else:
        print("Failed", testname, ";", msgFailed)
    return b

testif(input_tuple("Enter int, str, float separated by commas:", [int, str, float], ',')!=()\
       , 'test 1.1', 'ok', 'ko')
testif(input_tuple("Enter str, str separated by colons:", [str, str], ':')!=()\
       , 'test 1.2', 'ok', 'ko')
testif(input_tuple("Enter float, int, str, bool, separated by +:", [float, int, str, bool], '+')!=()\
       , 'test 1.3', 'ok', 'ko')

testif(input_tuple_lc("Enter float, int, str, bool, separated by +:", [float, int, str, bool], '+')!=()\
       , 'test 2.1', 'ok', 'ko')
```

```
testif(input_tuple_lc("Enter float, int, str, bool, separated by +:", [float, int, str, bool], '+')!=()\
      , 'test 2.2', 'ok', 'ko')
testif(input_tuple_lc("Enter float, int, str, bool, separated by +:", [float, int, str, bool], '+')!=()\
      , 'test 2.3', 'ok', 'ko')
```

Problem 2

```
#Michael Quach
import math
#Problem 2. Pythagorean Numbers Revisited
#Write a function named compute_Pythagoreans in file p2.py
#that takes one positive int argument n and returns a list
#with tuples (a,b,c), with 0<a,b,c<=n, such that a2+b2=c2.
#The function MUST use one list comprehension and no loops,
#or no credit is given for the solution.
#
#The function should also use proper error checking and
#return an empty list if the input parameter is invalid.

def compute_Pythagoreans(n):
    try:
        limit=int(n)
    except:
        print("Error: input must be a positive non-zero integer. Terminating.")
        return []

    a,b,c=1,1,1
    r=[(a,b,c) for a in range(limit) for b in range(limit) for c in range(limit)\
      if c==math.sqrt((a*a)+(b*b))]
    return r
```

Problem 3

I didn't get to this in time.