SELECTED SOLUTIONS

CHAPTER 5

Exercise Set 5.1.1

1. -1
2. 30
3. 180
4. -44
5. 4
6. ((-9)-((-4)**3-(-4)))*(-5)
7. ((90/5)**2-10)*11

Exercise Set 5.1.2

1. A. Real

   B. not Real

   C. Real, Rational, Integer

   D. Real, Rational

   E. Real, Rational

   F. Real, Rational, Integer, Whole Number

   G. Real

   H. Real, Rational

2. A. Answers will vary. Example: $\frac{7}{24}$

   B. Answers will vary. One technique is to recognize that $\frac{1}{4} = \sqrt{\frac{1}{16}}$ and $\frac{1}{3} = \sqrt{\frac{1}{9}}$ so choose a non-square number in between $\sqrt{\frac{1}{16}}$ and $\sqrt{\frac{1}{9}}$ such as $\sqrt{\frac{1}{10}}$.

3. $\frac{12}{99}$

4. $\frac{mn}{99}$

Exercise Set 5.1.3

1. The result is a float.
2. A. 3 B. 3 C. -3 D. -3
3. The result of **int**() is that it truncates the decimal portion of the number.
4. A. -4/3

   B. 42

   C. ½

   D. 442139859559509/140737488355328

   E. 1/8

   F. Type Error

   G. Type Error
5. A. 2

   B. 2.125

   C. 8782019273372467/2251799813685248

   D. 3.9
6. Yes, the result is a float.
7. Yes, the result is a Fraction.

Exercise Set 5.1.4

1. The program will return the numerator and denominator in simplest form.
2. Yes it will accept input of decimal numbers.
3. 3/10 * 1/2 = 3/20
4. Modify **multiply_fractions.py** so that it also divides the two fractions.

```
from fractions import *
```

```
F1 = Fraction(input('Fraction 1? '))
F2 = Fraction(input('Fraction 2? '))

a,b = F1.numerator, F1.denominator
c,d = F2.numerator, F2.denominator

e = a*c
f = b*d
F3 = Fraction(e,f)
print(F1, "*", F2, "=", F3)

g = a*d
h = b*c
F4 = Fraction(g,h)
print(F1, "divided by", F2, "=", F4)
```

5. Write a program that adds a pair of fractions.

```
from fractions import *
F1 = Fraction(input('Fraction 1? '))
F2 = Fraction(input('Fraction 2? '))

F3 = F1 + F2

print(F1, "+", F2, "=", F3)
```

6. Write a function that will convert a string fraction to a float variable rounded to 3 decimal places.

```
def convert_to_float(f):
    from fractions import Fraction
    f = Fraction(f)
    return round(float(f),3)
```

Exercise Set 5.2.1

1. A. -4**2 = -16 (int)
   B. -4**2 = 16 (int)
   C. 4.0**2 = 16.0 (float)
   D. -64**(1/2) = -8.0 (float)
   E. (-64)**(1/2) = (4.898587196589413e-16+8j) (float) (essentially -8i)
   F. Fraction(1,4)**2 = 1/16 (fraction)
   G. (1/4)**2 = .0625 (float)
   H. math.sqrt(-16) math domain error

I. pow(1/4,2) = .0625 (float)
J. 64**(1/3) = 3.9999999999999996 (float)

2. A. divmod(17,2) = (8,1) (tuple)
B. divmod(-17,2) = (-9,1) (tuple)
C. divmod(-17,-2) = (8,-1) (tuple)
D. divmod(17,-2) = (-9,-1) (tuple)
E. 17%2 = 1 (int)
F. 17%0 returns ZeroDivisionError

3. No, divmod(10,-4) = (-3, -2).

4. A. 0

B. 1

5. A. Does (a + b) % m = a % m + b % n? Yes
B. Does a % m + a % n = a % (m + n)? No

6. What snippet of code can gives the true value of a^b ?

```
# find a**b
if a>=0:
   return a**b
else
   return -abs(a)**b
```

7. $\frac{m-m\%n}{n}$

8. m%n = 0 if m|n.

9. Convert 12-hour time to 24-hour time.

```
hour = int(input("hour? "))
min = int(input("minutes? "))
ap = input("am or pm? ")

if hour==12:
   if ap == "am":
      hour24 = 0
   elif ap == "pm":
      hour24 = 12
else:
   if ap == "pm":
      hour24 = hour+12
   else:
      hour24 = hour
print(hour24,":",minutes)
```

10. Write a program that performs the same function as **divmod**.

```
m = int(input("dividend? "))
n = int(input("divisor? "))
r = m%n

q = int((m-r)/n)
print("Quotient:",q,"remainder:",r)
```

Exercise Set 5.2.2

1. A. abs(-3.8)
   B. round(-3.8)
   C. int(-3.8)
   D. math.floor(-3.8)
   E. math.floor(3.8)
   F. math.ceil(-3.8)
   G. math.ceil(3.8)
   H. math.trunc(-3.8)
   I. math.trunc(3.8)
   J. math.gcd(12,126)
   K. math.gcd(-12,-126)

2. The difference is how each function treats negative numbers. math.**floor**(-3.14) returns -3 (the greatest integer to the left). math.**trunc**(-3.14) returns -3 (the integer portion of the number).

3. Write a brief function that finds the decimal portion of a float number. For example, fractional_part(3.14159) = .14159.

```
def fractional_part(x):

   f = x - trunc(x)
   return f

# main program
from math import *
:
:
```

4. Define **round**() as a piecewise-defined function on a positive number using math.**floor**() and math.**ceil**().

```
def round(x):
   x = abs(x)
   dec_part = x - floor(x)
```

```
   if dec_part < 0.5:
      r = floor(x)
   else:
      r = ceil(x)
   return r

# main program
from math import *
:
:
```

5. No, **int**() and math.**trunc**() are essentially the same.

6. Write a program that determines if a positive number is a perfect square.

```
from math import sqrt
x = int(input("Positive integer? "))
if sqrt(x) == int(sqrt(x)):
   print(x,"is a perfect square of",int(sqrt(x)))
else:
   print(x,"is not a perfect square")
```

7. Two numbers are relatively prime if their gcd is 1.

8. No. Counterexample: if x = -7.6 and y = -5.4 then $\lfloor -7.6 \rfloor + \lfloor -5.4 \rfloor = -8 + -6 = -14$ but $\lfloor -7.6 + -5.4 \rfloor = \lfloor -13 \rfloor = -13$.

9. Write a program that expresses a positive rational number m/n as m = q·n + r, where m>n and 0 ≤ r < n.

```
F = input('Fraction? ')
m,n = F.split('/')
m,n = int(m),int(n)

r = m%n
if r==0:
   q = int(m/n)
   print(m, "=", q, "*", n)
else:
   q = int((m-r)/n)
   print(m, "=", q, "*", n, "+",r)
```

10. Write a program that asks the user to input values for k, m, and n. Calculate and output the number of multiples of k between m and n.

```
def multiples(k,m,n):
   m = int(n/k) - int((m-1)/k)
```

```
    return m

#main program
k = int(input("k? "))
m = int(input("m? "))
n = int(input("n? "))
num = multiples(k,m,n)
print("The number of multiples of",k,"between",m,"and",n,"is",num)
```

Exercise Set 5.2.3

1. No, you get 2 lb, 12.799999999999997 oz. This can be fixed by inserting above the **print** statement:

```
oz = round(oz,2)
```

2. Append the following lines to the program:

```
total_oz = 16*lb + oz
fee = round(0.68*total_oz,2)
print("The shipping charge is $",fee)
```

3. Import the **math** module and replace

```
oz = round(oz,2)
```

with:

```
oz = ceil(oz)
```

4. Modify the program so that $3.00 is charged for the first pound and 0.62 for each additional oz.

```
from math import *
weight = float(input("weight in lb? "))

lb = int(weight)
fractional_part = weight - lb
oz = fractional_part*16
oz = ceil(oz)
print("The package weight is", lb, "lb,", oz, "oz")
total_oz = 16*lb + oz
excess = total_oz - 16
cost = excess*0.62

print("The shipping charge is $", 3+cost)
```

Exercise Set 5.2.4

1. Write a program so that the user can input any positive integer and the result is the simplified positive square root.

```
from sympy import *
a = int(input("positive integer? "))
pprint(sqrt(a))
```

2. Modify the program so that it verifies that the input is a positive integer.

```
from sympy import *

a = int(input("positive integer? "))
if a >= 0:
   pprint(sqrt(a))
else:
   print("Not a real number")
```

3. A. $\sqrt{492} = 2\sqrt{123}$

   B. $\sqrt{3456} = 24\sqrt{6}$

   C. $\sqrt{2020} = 2\sqrt{505}$

   D. $\sqrt{4753} = 7\sqrt{97}$

4. Write a program to find the cube root of a number.

```
from sympy import *
x = Symbol('x')
a = input("Integer? ")
n = float(a)
if n != int(n):
   print(a, "is not an integer")
else:
   expr = "x**3-"+a
   roots = solve(expr)
   pprint(roots[0])
```

Exercise Set 5.3.1

1. A. 816

B. 495

C. 1364

2. The answer is positive if one of the inputs is negative; negative if both inputs are negative.

3. Python returns a ZeroDivisonError.

Insert above the print statement:

```
if m == 0 or n == 0:
  print("Error.  You can only find the lcm of nonzero numbers.")
else:
```

4. Rewrite lcm function.

```
def lcm(m,n):
   from math import gcd
   result= (m*n)/gcd(m,n)
   return result
:
:
```

5. The function returns a float because it is the result of a division. Replace **return** result with:

```
return int(result)
```

6. Modify the main program to find the lcm of three numbers.

```
:
:
# main program
a = int(input("First number? ")
b = int(input("Second number? ")
c = int(input("Third number? ")
if a==0 or b==0 or c==0:
   print("Error.  You can only find the lcm of nonzero numbers.")
else:
   d = lcm(a,b)
  print("\nlcm(",a,",",b,",",c,") =",lcm(c,d))
```

7. CHALLENGE: Write a program that will add fractions $\frac{a}{b} + \frac{c}{d}$ without using the **Fraction**() function.

```
from math import gcd

from fractions import Fraction

def lcm(m,n):
```

```
    return (m*n)/gcd(m,n)

F1 = input('Fraction 1? ')
a,b = F1.split('/')
a,b = int(a), int(b)

F2 = input('Fraction 2? ')
c,d = F2.split('/')
c,d = int(c), int(d)

L = lcm(b,d)
m = L/b
n = L/d

F3_n = int(a*m+c*n)
F3_d = int(L)
g = gcd(F3_n,F3_d)
F3_n = int(F3_n/g)
F3_d = int(F3_d/g)
print(F3_n, '/', F3_d)
print(int(a*m+c*n), '/', int(L))
```

Exercise Set 5.4.1

1. A. gcd(56,284) = 4
   B. gcd(77,91) = 7
   C. gcd(3725,4335) = 5
   D. gcd(4225,1625) = 325

2. Replace the **print**() statement with:

```
if m == 1:
   print("relatively prime")
else:
   print(m)
```

3. The program goes into an infinite loop if one of the inputs is 0. Suggested fix:

```
m = int(input("First number? ")
n = int(input("Second number? ")
if m == 0:
   print(n)
elif n == 0:
   print(m)
else:
```

```
print("gcd(",m,",",n,")=")
while m != n
  if m > n:
    m = m - n
  if n > m:
    n = n – m
print(m)
```

4. Just insert the following line after the first **print** statement:

```
m = abs(m)
n = abs(n)
```

5. gcd(m,2m+n) = gcd(m,n)

Exercise Set 5.5.1

1. Modify program to handle 7 columns per row.

```
def grid(num_list):
 from math import floor
 fstring=""
  for k in range(7):
    fstring += '{:^5s}'

dim = len(num_list)
num_rows = floor(dim/7)
for row in range(num_rows+1):
  if len(num_list)>=7:
      # assign to row 1st 7 entries in num_list to row_to_print
    row_to_print = num_list[:7]
      # remove 1st 7 entries from num_list
    num_list = num_list[7:]
      # print next row
    print(fstring.format(*row_to_print))
  else:
    last_fstring = ''
    for k in range(len(num_list)):
      last_fstring += '{:^5d}'
    print(last_fstring.format(*num_list))

#MAIN PROGRAM
:
```

:

2. Modify program to handle n columns per row

Change **def** grid(num_list) to **def** grid(num_list,n) and within the body of the function, replace every occurrence of 7 with n.

Exercise Set 5.5.2

1. Fill in yellow highlighted code:

grid(prime,6)

2. The list is missing the first prime, which is 2.

One possible fix is to change **for** i **in range**(0,n+1,2) to **for** i **in range**(4,n+1,2). Or add a line: potential[2]=1.

3. Add the following line at the end of the program:

print("Total:",len(prime),"primes <=",n)

4. Insert at the beginning of the main program:

from math import log,e

Append to the end of the main program:

print("Approximated:",round(n/**log**(n,e)))

The approximate number of primes less than 100 is 22.

5. One possible improvement is to replace **while** j<=n with **while** j<=int(n/2).

6. Modify the main program to determine if input m is prime.

Insert at the beginning of the main program:

```
m = int(input("m? ")
n = m
```

Replace the last two lines (**grid** and **print**) with the following:

```
if m in prime:
   print(m,"is prime")
else:
   print(m,"is not prime")
```

Exercise Set 5.5.3

1. Rewrite program **prime_check** as a main program that asks for the user to input positive integer n. Call function **prime_check**(n) to return true or false.

```
def prime_check(n):
   comp = False

   if n > 1:
   # check for factors
      for i in range(2, n):
         if (n % i) == 0:
            comp = True

            break
   return comp


# main program
n = int(input("Enter a positive integer: "))
prime = prime_check(n)

if prime:
   print(n, "is a prime number")
else:
   print(n, "is composite")
```

2. Print out the first p primes.

```
def prime_check(n):
   prime = True

   if n > 1:
   # check for factors
      for i in range(2, n):
         if (n % i) == 0:
            prime = False
           break
   return prime

# main program
p = int(input("How many primes do you want? "))

primes = [ ]
count = 0
n = 2
```

```
while len(primes) < p:
   if prime_check(n):
      primes.append(n)
   n += 1

print('The first', p, 'primes are:')
print(primes)
```

3. Output all primes less than M.

```
def prime_check(n):
   prime = True

   if n > 1:
   # check for factors
      for i in range(2, n):
         if (n % i) == 0:
            prime = False
            break
   return prime

# main program
M = int(input("Upper bound for primes? "))

primes = [ ]
n = 2
while n < M:
   if prime_check(n):
      primes.append(n)
   n += 1

print('The primes less than', M, 'are:')
for k in primes:
   print(k)
```

4. Generate a random list of numbers. Calculate the percent that are prime.

```
from random import randint

def prime_check(n):
   prime = True

   if n > 1:
   # check for factors
      for i in range(2, n):
         if (n % i) == 0:
            prime = False
```

```
            break
    return prime

# main program

n = int(input("How many random whole numbers? "))

primes = [ ]
numbers = [ ]
for i in range(n):
    d = randint(1,10000)
    numbers.append(d)
    if prime_check(d):
        primes.append(d)

print("The random numbers are:")
print(numbers)
print("The primes in that list are:")
print(primes)
print("The percentage of primes is",len(primes)/len(numbers))
```

The percentage of primes is usually between .10 and .20.

Exercise Set 5.6.1

1. Insert factors.sort() above the **print** statement.

2. Any number with a prime factor greater than 100.

3. Write a program that finds all integer factors of a given number.

```
n = int(input("n? "))
factors = [ ]                    # initialize list of factors
for d in range(1,int(n/2)):
    if n%d==0:                     # if d is a factor of n
        factors.append(d)

print(factors)
```

Exercise Set 5.6.2

1. Fill in the yellow highlighted code with the following:

```
fstring = ''
for k in range(len(unique_factors)):
```

```
    fstring += “(“ + str(each_prime[k]) + “^” + str(exponent[k]) + “)”
print(fstring)
```

2. Modify the program so that if the exponent of any prime factor is 1, the exponent is not output.

```
primes = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]
n = int(input("n? "))
factors = [ ]                          # initialize list of factors
while n > 1:
   for p in primes:
      if n%p==0:                       # if p is a factor of n
         factors.append(p)
         n = int(n/p)

factors.sort()                         # sorts the prime factors

unique_factors = set(factors)          # creates a set of unique prime factors
each_prime = [ ]
exponent = [ ]
for f in unique_factors:
   each_prime.append(f)
   exponent.append(factors.count(f))

fstring = ""
for k in range(len(unique_factors)):
   if exponent[k]==1:
      fstring += "(" + str(each_prime[k]) + ")"
   else:
      fstring += "(" + str(each_prime[k]) + "^" + str(exponent[k]) + ")"
print(fstring)
```

3. A. $2^3 \cdot 3^5 \cdot 5 \cdot 11^2$

B. $5^3 \cdot 7^2 \cdot 13^3$

C. $3^5 \cdot 11^3 \cdot 17$

D. $37^2 \cdot 73^3$

Exercise Set 5.6.3

1. 198 does not produce a result because 198 = 5 + 193, and 193 was not part of the **primes** list we declared.

2. Modify program so that each prime pair only output once.

```
primes = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]
```

```
n = int(input("Positive Even integer? "))
i = 0
halfway= int(n/2)
while i < len(primes) and primes[i] <= halfway:
   for p2 in primes:
      if primes[i] + p2==n:
         print(n,"=",primes[i],"+",p2)
         break                              # Exit inner loop  when answer found
   i += 1
print("done")
```

3. 97 + 97 = 194

4. Insert the following after the input statement:

```
if n%2 != 0 or n <= 2:
   print("n must be positive, even and greater than 2")
else:
```

5. Modify program **goldbach** to implement Vinogradov's Theorem including checks that the input is a "sufficiently large" positive odd integer.

```
p = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83.89,97]
n = int(input("positive odd integer? "))
if ((n<7)or(n>291)):
   print("out of range")
elif (n%2 ==0):
   print("must be odd")
else:
   triples = []
   for p1 in p:
      for p2 in p:
         for p3 in p:
            if p1+p2+p3==n:
               if sorted([p1,p2,p3]) not in triples:
                  triples.append(sorted([p1,p2,p3]))
                  break
for k in triples:
   print(k)
```

6. Write a program to find all Mersenne Primes less than 10,000.

```
def prime_check(n):
   # function that returns True if n is prime; False otherwise
    prime = True
```

```
    if n > 1:

        for i in range(2, n):
            if (n % i) == 0:
                prime = False
                break
    return prime

def prime_generator(m):
    # function that generates a list of primes less than or equal to m
    primes = []

    for i in range(2,m+1):
        prime = prime_check(i)
        if prime:
            primes.append(i);
    return primes

# main program
primes = prime_generator(10000)
mersenne = [ ]                          # initialize list of Mersenne primes
k = 2
m = 3
while m <= 10000:
    m = 2**k-1
    if m in primes:
        mersenne.append(m)
    k +=1

print(mersenne)
```

The only Mersenne primes less than 10,000 are 3, 7, 31, 127, 8191.

7. Write a program to find all Twin Primes less than 500.

```
def prime_check(n):
    # function that returns True if n is prime; False otherwise
    prime = True

    if n > 1:

        for i in range(2, n):
            if (n % i) == 0:
                prime = False
                break
    return prime
```

```
def prime_generator(m):
   # function that generates a list of primes less than or equal to m
   primes = [ ]

   for i in range(2,m+1):
      prime = prime_check(i)
      if prime:
         primes.append(i);
   return primes

# main program
primes = prime_generator(500)
twin_primes = [ ]

for p in primes:
   if p+2 in primes:
      twin_primes.append([p,p+2])


print(twin_primes)
```

The twin primes less than 500 are:

[[3, 5], [5, 7], [11, 13], [17, 19], [29, 31], [41, 43], [59, 61], [71, 73], [101, 103], [107, 109], [137, 139], [149, 151], [179, 181], [191, 193], [197, 199], [227, 229], [239, 241], [269, 271], [281, 283], [311, 313], [347, 349], [419, 421], [431, 433], [461, 463]]

8. Write a program that asks the user to input n. Find p that satisfies Bertrand's Postulate.

```
def prime_check(n):
   # function that returns True if n is prime; False otherwise
    prime = True

    if n > 1:

       for i in range(2, n):
          if (n % i) == 0:
             prime = False
             break
   return prime

def prime_generator(m):
   # function that generates a list of primes less than or equal to m
   primes = []

   for i in range(2,m+1):
      prime = prime_check(i)
```

```
      if prime:
         primes.append(i);
   return primes

# main program
primes = prime_generator(1000)

n = int(input("positive integer n? "))
for p in primes:
   if ((p>n)and(p<=2*n)):
      print(p,"is between",n,"and",2*n)
      break;
```

Exercise Set 5.7.1

1. T and T → T

2. T and T → T

3. F or T → T

4. F or T → T

5. (T or F) and T → T

Exercise Set 5.7.2

1. ¬T → F

2. ¬ (T V F) → ¬T → F

3. ¬F ^ ¬F → T ^ T → T

4. ¬F V ¬F → T V T → T

5. T ^ ¬ (T V F) → T ^ T → T

6. The statement is True.

P	Q	P ^ Q	¬P	¬Q	¬P V ¬Q	¬ (¬P V ¬Q)
T	T	T	F	F	F	T
T	F	F	F	T	T	F
F	T	F	T	F	T	F
F	F	F	T	T	T	F

7. The statement is True.

P	Q	R	Q V R	P ^ (Q V R)	P ^ Q	P ^ R	(P ^ Q) V (P ^ R)
T	T	T	T	T	T	T	T
T	T	F	T	T	T	F	T
T	F	T	T	T	F	T	T
T	F	F	F	F	F	F	F
F	T	T	T	F	F	F	F
F	T	F	T	F	F	F	F
F	F	T	T	F	F	F	F
F	F	F	F	F	F	F	F

8. T ^ ¬F → T

9. T ^ F → F

10. T ^¬F → T

11. T ^ F → F

12. Possible Answer: ((A = 0) ^ ¬ (B = 0))) V ((¬ (a = 0) ^ (b = 0))

13. Possible Answer: (**abs**(f - L) > e) ^ (e > 0) ^ (e < 1)

Exercise Set 5.7.3

1. The result is 0 and the variable type is int.

2. The result is False and the variable type is bool.

3. The result is True and the variable type is bool.

4. The result is True and the variable type is bool.

Exercise Set 5.7.4

1. value[0] = 'F' and value[1] = 'T'

2. p and q are both bool type.

3. p **or** q = True and value[p **or** q] = 'T'.

4. Yes that is exactly what is happening with the list value.

5. Fill in code to output remaining four columns

```
print('{:^5s}{:^5s}{:^7s}{:^7s}{:^5s}{:^5s}{^7s}'.format(value[p],value[q],value[p and q],value[not (p
        and q)],value[not p],value[not q],value[(not p) or (not q)]))
```

6. Modify the program so that it generates and outputs 2nd version of DeMorgan's Formula.

7. A. Write a program that generates the truth table for p **xor** q.

B. Symbolically: (P ^ ¬Q) v (¬P ^ Q). The truth table is:

P	Q	P XOR Q
0	0	0
0	1	1
1	0	1
1	1	0

Exercise Set 5.8.1

1. Yes, primitive.

2. Yes, not primitive.

3. No.

4. Yes, primitive.

5. Yes, not primitive.

6. True. Example: (9, 40, 41)

7. True. Example: (9, 40, 41)

8. True. Example: (44, 117, 125). There is only one prime factor of 125:

5 = 4(1) + 1

9. True. Example: (8, 15, 17). The area is ½ (8)(15) = 60 which is divisible by 6.

10. Write a program that determines whether three positive integers form a Pythagorean Triple, and if so, whether it is primitive.

```
from math import gcd
a_b_c_str = input("a,b,c? ")
a_b_c = a_b_c_str.split(",")          # create a list of the 3 numbers
triple = [int(k) for k in a_b_c]       #  convert the 3 numbers from string to int
triple.sort()
a,b,c = [k for k in triple]
if a**2 + b**2 == c**2:
   print(a,b,c, "form a Pythagorean triple")
   if gcd(a,b) == 1 and gcd(b,c)==1 and gcd(a,c)==1:
      print("which is primitive")
else:
   print(a,b,c, "do not form a Pythagorean triple")
```

Exercise Set 5.8.2

1. A. 85, 132, 157

   B. 195, 748, 773

   C. 232, 825, 857

2. Write a program that asks the user to input two positive integers in which m > n. Generate a Pythagorean Triple based on these numbers. Make sure your program checks that inputs m and m are both positive integers and m > n.

```
from math import gcd
m_n_str = input("m,n? ")
m_n = m_n_str.split(",")
m_n = [abs(int(k)) for k in m_n]

m = m_n[0]
n = m_n[1]

if n > m:                                  # if n > m then swap them
   m,n = n,m

a = m**2 - n**2
b = 2*m*n
c = m**2 + n**2

if gcd(m,n)==1:

   if ((m % 2 ==1) and (n % 2==1)):        #  if m and n both odd
      a = a/2
```

```
    b = b/2
    c = c/2
print(int(a),int(b),int(c))
```

3. Is it possible to generate a primitive Pythagorean triple if m and n are not relatively prime? If so, how? Modify the program accordingly.

Yes, just divide m and n by their gcd to make them relatively prime.

```
from math import gcd
m_n_str = input("m,n? ")
m_n = m_n_str.split(",")
m_n = [abs(int(k)) for k in m_n]

m = m_n[0]
n = m_n[1]

if n > m:                              # if n > m then swap them
   m,n = n,m

   if gcd(m,n)!=1:                     # if m, n not relatively prime
      g = gcd(m,n)
      m = int(m/g)
      n = int(n/g)

a = m**2 - n**2
b = 2*m*n
c = m**2 + n**2

if gcd(m,n)==1:

   if ((m % 2 ==1) and (n % 2==1)):    #  if m and n both odd
      a = a/2
      b = b/2
      c = c/2

print(int(a),int(b),int(c))
```