SELECTED SOLUTIONS

CHAPTER 8

Exercise Set 8.1.1

1. zeros: 3 (multiplicity 3, crosses); -2 (multiplicity 2, touches); -5 (multiplicity 4, touches)

2. zeros: 0 (multiplicity 1, crosses); -4 (multiplicity 2, touches)

3. zeros: 2 (multiplicity 2, touches)

4. zeros: -3 (multiplicity 1, crosses); 3 (multiplicity 1, crosses); 1 (multiplicity 1, crosses)

Exercise Set 8.1.2

1. Fill in yellow highlighted code to indicate whether the graph touches or crosses at each zero.

```
    z = zeros[k]
   m = mults[k]
   message = " at z="+str(z)
   if m%2==0:
      message ="touches"+message
   else:
      message = "crosses"+message
   print(message)
```

2. Modify the program so that it calculates and outputs the degree of the expanded polynomial.

```
zeros = [ ]
mults = [ ]
pairs_list = input("Enter Zero, multiplicity pairs: ")
pairs = pairs_list.split(';')
n = len(pairs)
for p in pairs:
   zero_mult = p.split(',')
   zeros.append(zero_mult[0])
   mults.append(zero_mult[1])

degree = 0
for k in range(n):
   z = zeros[k]
```

```
    m = int(mults[k])
    degree += m
    message = " at z="+str(z)
    if m%2==0:
        message ="touches"+message
    else:
        message = "crosses"+message
    print(message)

print("degree:",degree)
```

3. Streamline the data entry for the user so that the data can be done with a single input statement.

```
zeros = [ ]
mults = [ ]
pairs_list = input("Enter Zero, multiplicity pairs: ")
pairs = pairs_list.split(';')
n = len(pairs)
for p in pairs:
    zero_mult = p.split(',')
    zeros.append(zero_mult[0])
    mults.append(zero_mult[1])

for k in range(n):
    z = zeros[k]
    m = int(mults[k])
    message = " at z="+str(z)
    if m%2==0:
        message ="touches"+message
    else:
        message = "crosses"+message
    print(message)
```

4. Modify the program so that it also outputs the polynomial in factored form.

```
from sympy import Symbol,sympify,pprint
x = Symbol('x')
:
:
poly = ""
for k in range(n):
    z = int(zeros[k])
    m = int(mults[k])
    if z>0:
        factor = "*((x-"+str(z)+")"
    else:
        factor = "*((x+"+str(-z)+")"
```

```
    factor += "**"+str(m)+")"
    poly += factor
# remove the first *
poly = poly[1:]
poly=sympify(poly)
pprint(poly)
```

Exercise Set 8.1.3

1. A. $(x + 1)^2(x – 2)^2$
Zeros: -1, multiplicity 2; 2, multiplicity 2

B. $(x - 3)(x + 7)^2$
Zeros: 3, multiplicity 1; -7, multiplicity 2

C. $(x + 5)^2(x – 3)(x + 1)^3$
Zeros: -5, multiplicity 2; 3, multiplicity 1; -1, multiplicity 3

2. Type and value of each of the following variables.

Variable	Type	Value
factor_set	tuple	(1, [(x + 5, 2)])
f_list	list	[(x + 5, 2)]
f	tuple	(x + 5, 2)
f[0]	sympy	x + 5
fact_str	string	'x + 5'
z	list	[-5]

3. Replace the last line with:

```
    if mult % 2==0:
        description=',touches'
    else:
        description=',crosses'
    print(z,'has multiplicity',mult,description)
```

and adjust indentation of following lines accordingly.

4. Insert the following line above mult=f[1]:

```
if len(z)==1:
```

5. Use the functions in poly_fcn.py so user can input coefficients of the polynomial rather than the entire polynomial.

```
from sympy import factor_list,factor,pprint,solve
import poly_fcn as pf
```

```
poly_coeff=input('Coefficients of Polynomial Function? ')
coeff = pf.num_string_to_list(poly_coeff)
poly_str = pf.num_list_to_poly(coeff)

factor_set = factor_list(poly_str)
factored_form = factor(poly_str)
:
:
```

Exercise Set 8.2.1

1. $(3x - 5)(x - 4)(x - 11)$

2. $(x - 5)(x + 4)^2(3x + 1)$

3. $(4x + 1)(2x + 1)(x - 2)(x + 3)(x - 4)$

Exercise Set 8.2.2

1. $(3x + 2)(x - 3)(x^2 + 5)$

2. $(x + 4)^2(x - 5)(x^2 + 3)$

3. $(x + 2)(x - 3)(4x^2 + x + 1)$

Exercise Set 8.2.3

1. Two rational zeros

2. No real zeros

3. One real zero

4. No real zeros

5. Write a program that uses the discriminant to identify the number and type of solutions for quadratic equation $ax^2 + bx + c = 0$.

```
from math import sqrt
coeff=input("Coefficients a,b,c? ")
coeff_list=coeff.split(',')
```

```
a,b,c = [int(coeff_list[i]) for i in range(3)]
d = b**2-4*a*c
print("discriminant: ",d)
if d == 0:
   print("one rational solution")
elif d < 0:
   print("no real solutions")
else:   # d is positive
   if sqrt(d) == int(sqrt(d)):  # d is a perfect square
      print("Two rational solutions")
   else:
      print("Two irrational solutions")
```

Exercise Set 8.2.4

1. **Fraction**(1/3) returns 6004799503160661/18014398509481984 and **Fraction**(1,3) returns 1/3, and **Fraction**(1.0,3.0) returns 'TypeError.'

2. Modify the program to handle all sample output correctly.

```
from math import sqrt
from fractions import Fraction

coeffs = input("a,b,c? ")
a,b,c = coeffs.split(',')
a,b,c = float(a),float(b),float(c)

if b**2-4*a*c < 0:
   print("no real solution")
else:
   x1_num = -b + sqrt(b**2-4*a*c)
   x2_num = -b - sqrt(b**2-4*a*c)
   den = 2*a

   x1 = Fraction(x1_num/den)
   x1 = x1.limit_denominator(100)
   x2 = Fraction(x2_num/den)
   x2 = x2.limit_denominator(100)

   print(x1,",",x2)
```

3. Replace the a,b,c assignment statement with:

a,b,c = **Fraction**(a),**Fraction**(b),**Fraction**(c)

4.. Replace the **print** statement with the following:

```
if x1==0:
   factor1="x"
elif x1>0:
   factor1="(x-"+str(x1)+")"
else:
   factor1="(x+"+str(abs(x1))+")"
if x2==0:
   factor2="x"
elif x2>0:
   factor2="(x-"+str(x2)+")"
else:
   factor2="(x+"+str(abs(x2))+")"

factored = factor1+"*"+factor2
print(factored)
```

5. Use a function to eliminate repetitive code.

```
from math import sqrt
from fractions import Fraction

def form_factor(n,d):
   x = Fraction(n/d)
   x=x.limit_denominator(100)
   if x==0:
      factor="x"
   elif x>0:
      factor="(x-"+str(x)+")"
   else:
      factor="(x+"+str(abs(x))+")"
   return factor

coeffs = input("a,b,c? ")
a,b,c = coeffs.split(',')
a,b,c = Fraction(a),Fraction(b),Fraction(c)

if b**2-4*a*c >=0:
   x1_num = -b + sqrt(b**2-4*a*c)
   x2_num = -b - sqrt(b**2-4*a*c)
   den = 2*a

   factor1 = form_factor(x1_num,den)
   factor2 = form_factor(x2_num,den)
```

```
    factored = factor1+"*"+factor2
    print(factored)
else:
    print("no real solution")
```

6. What factorization is returned for $6x^2 + 19x + 15$? What is the actual factorization?

Returned: (x+3/2)(x+5/3)

Actual: (2x + 3)(3x + 5)

Exercise Set 8.2.5

1. Write a function that finds the simplified square root of a positive integer.

```
# Main program must include sympy

  def simplified_radical(n):

    sq_rt = sympy.sqrt(n)

    sq_rt_str=str(sq_rt)
    sq_rt_str = sq_rt_str.replace('sqrt','\u221a')
    sq_rt_str = sq_rt_str.replace('*','')
    return sq_rt_str
```

2. Modify program **quad_root** so that it detects irrational zeros and outputs the associated factors in simplified radical form.

```
import math
import sympy
from fractions import Fraction

def form_factor(n,d):
    x = Fraction(n/d)
    x=x.limit_denominator(100)
    if x==0:
        factor="x"
    elif x>0:
        factor="(x-"+str(x)+")"
    else:
        factor="(x+"+str(abs(x))+")"
    return factor

coeffs = input("a,b,c? ")
```

```
a,b,c = coeffs.split(',')
a,b,c = Fraction(a),Fraction(b),Fraction(c)

if b**2-4*a*c >=0:
   x1_num = -b + math.sqrt(b**2-4*a*c)
   x2_num = -b - math.sqrt(b**2-4*a*c)
   den = 2*a

   if math.sqrt(b**2-4*a*c)== int(math.sqrt(b**2-4*a*c)):
   # discriminant is a perfect square
      factor1 = form_factor(x1_num,den)
      factor2 = form_factor(x2_num,den)
      print(factor1+"*"+factor2)
   else: # discriminant is not a perfect square

      zero1 = str(-b/(2*a)+ sympy.sqrt(b**2-4*a*c)/(2*a))
      zero2 = str(-b/(2*a)-sympy.sqrt(b**2-4*a*c)/(2*a))
      # replace 'sqrt' with unicode square root symbol
      zero1 = zero1.replace('sqrt','\u221a')
      zero2 = zero2.replace('sqrt','\u221a')
      # form factors
      factor1 = "[x-("+zero1+")]"
      factor2 = "[x-("+zero2+")]"
      factored = factor1+'*'+factor2
      print(factored)
else:
   print("no real solution")
```

3. Modify program to use **sympy.solve** to find zeros.

```
from fractions import Fraction
from sympy import solve

def form_factor(x):
   x_str = str(x)
   x_str = x_str.replace('sqrt','\u221a')
   if x==0:
      factor="x"
   else:
      factor="[x-("+x_str+")]"
   return factor

coeffs = input("a,b,c? ")
a,b,c = coeffs.split(',')
a,b,c = Fraction(a),Fraction(b),Fraction(c)

quadratic = str(a)+'*x**2+'+str(b)+'*x+'+str(c)
```

```
zeros=solve(quadratic)

# form factors
factor1 = form_factor(zeros[0])
factor2= form_factor(zeros[1])
factored = factor1+'*'+factor2
print(factored)
```

4. Modify program to express repeated factors using an exponent.

```
from fractions import Fraction
from sympy import solve

def form_factor(x):
    x_str = str(x)
    x_str = x_str.replace('sqrt','\u221a')
    if x==0:
        factor="x"
    else:
        factor="[x-("+x_str+")]"
    return factor

coeffs = input("a,b,c? ")
a,b,c = coeffs.split(',')
a,b,c = Fraction(a),Fraction(b),Fraction(c)

quadratic = str(a)+'*x**2+'+str(b)+'*x+'+str(c)
zeros=solve(quadratic)

# form factors
factor1 = form_factor(zeros[0])
if len(zeros)==2:
    factor2= form_factor(zeros[1])
    factored = factor1+'*'+factor2
else:
    factored=factor1+'\u00b2'
print(factored)
```

Exercise Set 8.2.6

1. Insert the following lines above **print**(factored):

```
if a != 1:
    factored=str(a)+factored
```

2. Given a = 1, b = 0, c = -4, give the value and type of each of the following expressions.

A. 1*x**2+0*x+-4, string

B. x**2 - 4, sympy expression

C. x**2 – 4, string

D. x^2 – 4, string

3. Adapt program **quad_root** as a function.

```
from sympy import solve,sympify

def form_factor(x):
   x_str = str(x)
   x_str = x_str.replace('sqrt','\u221a')
   if x==0:
      factor="x"
   else:
      factor="[x-("+x_str+")]"
   return factor

def quad_factor(a,b,c):
   quadratic = str(a)+'*x**2+'+str(b)+'*x+'+str(c)
   if b**2-4*a*c >=0:
      zeros=solve(quadratic)
      # form factors
      factor1 = form_factor(zeros[0])
      if len(zeros)==2:
         factor2= form_factor(zeros[1])
         return factor1+'*'+factor2
      else:
         factor1=factor1+'\u00b2'
         return factor1
   else:  # discriminant < 0; nonreal zeros
      quadratic = sympify(quadratic)
      quadratic = str(quadratic)
      quadratic = quadratic.replace('**2','\u00b2')
      quadratic = quadratic.replace('*','')
      return '['+quadratic+']'
```

Exercise Set 8.3.1

1. Dividend: $x^3 – 4x^2 – 17x + 60$; Divisor: x-3; Quadratic Quotient: $x^2 – x - 20$; Factored Quotient: (x – 3)(x+4)(x – 5)

2. Insert code to output result.

```
print(factored_cubic)
```

3. The factorization is:

6*(x+2)*[x-(-1/2)]*[x-(4/3)])

4. Include at the beginning:

```
from fractions import Fraction
```

Then replace assignment statements (including z = 3) with input statements:

```
coeffs = input("Coefficients a,b,c,d? ")
a,b,c,d = [Fraction(k) for k in coeffs.split(',')]
z = Fraction(input("Zero? "))
```

5. Modify the program so that it also outputs the original polynomial.

Include at the beginning:

```
from sympy import sympify
```

Then insert the following code above **for** col ….

```
poly = str(a)+'*x**3+'+str(b)+'*x**2+'+str(c)+'*x+'+str(d)
poly=sympify(poly)
pprint(poly)
```

6. Verify that z is actually a zero of the given polynomial.

Insert the following lines of code above result = ….:

```
if int(row3[degree])!=0:
   print(z," is not a factor of",poly)
else:
```

Indent the following lines accordingly.

Exercise Set 8.3.2

1. ±1, ±1/2, ±1/3, ±1/6, ±2, ±2/3, ±7, ±7/2, ±7/3, ±7/6, ±14, ±14/3

2. ±1, ±1/2, ±1/3, ±1/4, ±1/6, ±1/12, ±5, ±5/2, ±5/3, ±5/4, ±5/6, ±5/12, ±7, ±7/2, ±7/3, ±7/4, ±7/6, ±7/12, ±35, ±35/2, ±35/4, ±35/6, ±35/12

3. ±1, ±1/2, ±2, ±4, ±8, ±16, ±32, ±64

4. No. The zero could be an irrational number or perhaps no real solution at all.

Exercise Set 8.3.3

1. Replace the yellow highlighted code to generate list q, factors of the first term.

```
q = [ ]
for k in range(1,first+1):
  if first % k==0:                    # if k is a multiple of first
    q.append(k)
```

2. No it is not necessary to iterate through range(1,n). There are no factors of n between n/2 and n, so one can use range(1,round(n/2)+1).

3. Insert the following line above the **print** statement:

pq_s.**sort**()

4. Revise function **pq** so that duplicates are eliminated.

Replace possible = p_over_q with:

possible = **set**(p_over_q)

5. Revise the program to include negative entries.

Add the following line after p.**append**(k):

p.**append**(-k)

Exercise Set 8.3.4

1. A. ±1, ±1/3, ±1/6, ±1/18, ±2, ±2/3, ±1/9, ±3, ±6

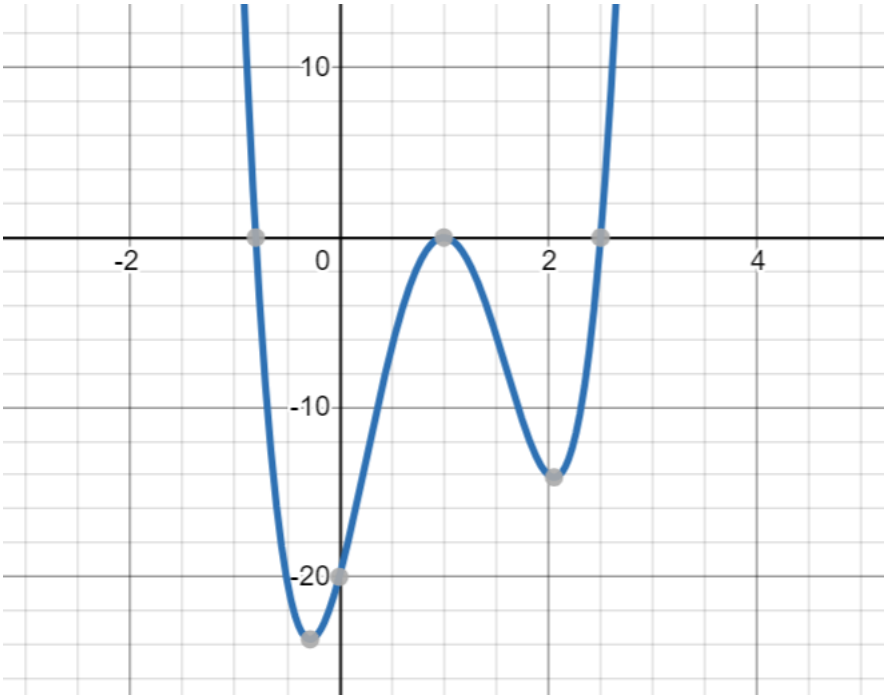
B. No

C. No

D. Yes

2. A. Possible Rational Zeros: ±1, ±1/2, ±1/5, ±1/10, ±2, ±2/5, ±4, ±4/5, ±5, ±5/2, ±10, ±20

Graph:



The most likely zeros are -1/2, -4/5,1,5/2

B. No.

C. Yes, because -4/5 is a zero.

3. Write a program to determine if a possible zero is an actual zero of a given polynomial.

```
from sympy import sympify,div

f = input('polynomial dividend?')
f = sympify(f)
z = input('possible zero? ')
g = 'x-'+z
g.replace('--','+')

q, r = div(f,g,domain='QQ')


if r ==0:
   print(z,'is a zero of',f)
else:
   print(z,'is not a zero of',f)
```

Exercise Set 8.4.1

1. $\sqrt{-120} = 2i\sqrt{30}$

2. x = $\pm\sqrt{-80} = \pm 4i\sqrt{5}$

3. (x + 9i)(x – 9i)

4. -1 ± 2i

5. $-\frac{3}{4} \pm \frac{\sqrt{39}}{4} i$

6. solve results in [-3*I, 3*I].

   Factor results in x**2+9.

7. Modify function **quad_factor** so that it will factor quadratics with nonreal zeros.

```
def quad_factor(a,b,c):
  quadratic = str(a)+'*x**2+'+str(b)+'*x+'+str(c)

  zeros=solve(quadratic)
  # form factors
  factor1 = form_factor(zeros[0])
  if len(zeros)==2:
    factor2= form_factor(zeros[1])
    return factor1+'*'+factor2
  else:
    factor1=factor1+'\u00b2'
    return factor1
```

Exercise Set 8.4.2

1. Write a program that adds a pair of complex numbers.

```
z1 = complex(input("z1? "))
z2 = complex(input("z2? "))
z3 = z1 + z2
print(z1,'+',z2,'=')
print(z3)
```

2. A complex type is returned.

Exercise Set 8.4.3

1. Modify program from Exercise Set 8.4.2 #1 so that user can input square root expressions.

```
from sympy import simplify,pprint

z1 = simplify(input("z1? "))
z2 = simplify(input("z2? "))
z3 = simplify(z1 + z2)

pprint(z3)
```

2. Modify the program so that the sum is expressed in standard form a + bi.

```
from sympy import simplify,pprint,re,im

z1 = simplify(input("z1? "))
z2 = simplify(input("z2? "))
z3_real = re(z1)+re(z2)
z3_imag = im(z1)+im(z2)
z3_str = str(z3_real)+'+('+str(z3_imag)+')\u2148'

print(z3_str)
```

Exercise Set 8.5.1

1. $i^{93} = i^{92} \cdot i = 1 \cdot i = i$

2. 11 + i

3. -8 – 7i

4. The result is (4.408109496293883e-15+1j). The actual answer is 0+1j which is simply i.

5. Write a program that will accept positive integer N as input and then simplify and output i^N.

```
n = int(input('n? '))
r = n % 4

if r ==0:
   p = '1'
if r==1:
   p = 'i'
if r==2:
   p = '-1'
if r==3:
   p = '-i'
```

```
print('i^',n,'=',p)
```

Exercise Set 8.6.1

1. 24 – 30i
2. 14 + 10i
3. 74 + 27i
4. 29
5. 192 + 56i
6. 18 – 26i
7. (6+5j)*(4+7j)

   The value returned is (-11+62j).

8. **simplify**((6+5*I)*(4+7*I))

   Assuming you preceded the command with:

   from sympy **import** *

   The value returned is -11 + 62*I.

Exercise Set 8.6.2

1. $7^2 + 8^2 = 113$
2. $1^2 + 10^2 = 101$
3. (x + 9i)(x – 9i)
4. (11x + 7i)(11x – 7i)
5. $5(3x + \sqrt{10}i)\ (3x - \sqrt{10}i)$
6. A. (x - 5)*(x + 5)

   B. x**2 – 24

   C. x**2 + 25

7. Write a program that will factor a sum of squares.

```
from sympy import *
poly = input('Sum of squares in the form x**2 + a? ')
terms = poly.split('+')
d = sqrt(int(terms[1]))          # convert the 2nd term to an integer and take square root
d = str(d)                       # convert d to a string so you can do string replacement
d = d.replace('sqrt','\u221a')   # replace sqrt with the radical sign
print('(x-',d,'i)(x+',d,'i)')
```

8. Write a brief program that finds the conjugate of a built-in complex type number.

```
z1 = complex(input("Complex Number? "))
z2 = z1.conjugate()
print("The conjugate is",z2)
```

9. Write a brief program that finds the conjugate of a **SymPy** complex type number.

```
from sympy import simplify, conjugate
z1 = simplify(input("Complex Number? "))
z2 = conjugate(z1)
print("The conjugate is",z2)
```

Exercise Set 8.7.1

1. $-\frac{3i}{7}$

2. -1 – *i*

3. $-\frac{12}{5}-\frac{4}{5}i$

4. $-\frac{5}{2}-\frac{1}{2}i$

5. -*i*

6. Write a program that finds the quotient of Complex numbers.

```
from sympy import *
x = Symbol('x')
z1 = sympify(input("Enter dividend: "))
z2 = sympify(input("Enter divisor: "))

quo = simplify(z1/z2)
pprint(quo)
```

7. Yes the program will also find the quotient of polynomials.

Exercise Set 8.8.1

1. $(x- (7+2i))(x-(7-2i)) = x^2 - 14x + 53$

2. $(x - 4)(x - (3 + i))(x - (3 - i)) = x^3 - 10x^2 + 34x - 40$

3. $x(x - 1)(x - 2)(x - i)(x + i) = x^5 - 3x^4 + 3x^3 - 3x^2 + 2x$

4. $(x - i)(x + i)(x - (1 + i))(x - (1 - i)) = x^4 - 2x^3 + 3x^2 - 2x + 2$

5. $(x - 3)(x - (4 + i))(x - (4 - i)) = x^3 - 11x^2 + 41x - 51$

6. $(x^2 + 9)(x -5)^2 = x^4 - 10x^3 + 34x^2 - 90x + 225$

7. Expand $(x - (a + bi))(x - (a - bi)) = (x - a - bi)(x - a + bi) = ((x-a)-bi)((x-a)+bi)$

   $= (x- a)^2 - (bi)^2 = x^2 - 2ax + a^2 - b^2i^2 = x^2 - 2ax + a^2 - b^2(-1) = x^2 - 2ax + a^2 + b^2$.

8. Write a program that expands $(x - (a + bi))(x - (a - bi)) = x^2 - 2ax + (a^2 + b^2)$.

```
a = int(input('a? '))
b = int(input('b? '))
print('(x-(', a, '+', b, 'i))(x -(', a, '-', b, 'i))=')
print('x**2 +', -2*a, 'x +', a**2+b**2)
```

Exercise Set 8.8.2

1. Include the following code to calculate and output the expanded quadratic:

```
factor1 = simplify(factor1)
factor2 = simplify(factor2)
poly=expand(factor1*factor2)
pprint(poly)
```

2. Modify the program so that the user can enter any additional zeros of the function.

```
from sympy import *

z1 = simplify(input("Complex Number? "))

z2 = conjugate(z1)

x = Symbol('x')
factor1 = '(x-('+str(z1)+'))'
factor2 = '(x-('+str(z2)+'))'

other_zs = input("other zeros? ")
```

```
other_factors = [ ]
if other_zs != "":
   zs = other_zs.split(',')              # zs =  list of other zeros
   for i in range(len(zs)):              # other_factors = list of (x-a) factors formed from zs
      if float(zs[i])>0:
         other_factors.append("(x-"+zs[i]+")")
      elif float(zs[i])<0:
         other_factors.append("(x+"+str(abs(float(zs[i])))+")")
      else:
         other_factors.append("x")

factor_list=factor1+factor2             # initialize factor_list with the conjugate pair
for i in range(len(other_factors)):      # for each factor in other_factors list
   factor_list+=other_factors[i]         # append to factor_list
print(factor_list,'=')

factor1 = simplify(factor1)             # simplify and multiply the conjugate pair
factor2 = simplify(factor2)
poly = expand(factor1*factor2)

factors=[ ]
for i in range(len(other_factors)):            # for each other factor
   next_factor = simplify(other_factors[i])    # convert to SymPy expression
   poly = expand(poly*next_factor)             # and multiply by previous poly

pprint(poly)
```

Exercise Set 8.8.3

1. $(x^2 + 72) = (x + 6\sqrt{2}i)(x - 6\sqrt{2}i)$. The zeros are $\pm 6\sqrt{2}$I (both imaginary).

2. The zeros of $x^2 + 7x + 13 = 0$ are $-\frac{7}{2} \pm \frac{\sqrt{3}}{2}i$ $(both\ imaginary)$. The factorization is $(x - (-\frac{7}{2} + \frac{\sqrt{3}}{2}i))(x - (-\frac{7}{2} - \frac{\sqrt{3}}{2}i))$.

3. $x^3 - 6x^2 - 31x + 36 = (x + 4)(x - 1)(x - 9)$. The zeros are -4, 1, 9 (all rational).

4. $x^3 + 81 = (x + 3)(x^2 - 3x + 9) = (x + 3)(x - (\frac{3}{2} + \frac{3\sqrt{3}}{2}i))\ (x - (\frac{3}{2} - \frac{3\sqrt{3}}{2}i))$. The zeros are -3 (rational), $\frac{3}{2} \pm \frac{3\sqrt{3}}{2}$i (imaginary).

5. $2x^3 - 3x^2 - 12x + 20 = (x-2)^2(2x + 5)$. The zeros are 2 (multiplicity 2) and -5/2 (all rational).

6. $x^4 + x^3 - 10x^2 - 4x + 24 = (x - 2)^2(x + 2)(x + 3)$. The zeros are 2 (multiplicity 2), -2, -3 (all rational).

7. $x^4 + x^3 - 2x^2 + 4x - 24 = (x - 2)(x + 3)(x^2 + 4) = (x - 2)(x + 3)(x - 2i)(x + 2i)$. The zeros are 2, -3 (rational) and $\pm 2i$ (imaginary).

8. $x^4 + 4 = (x^2 - 2x + 2)(x^2 + 2x + 2) = (x - (1+i))(x - (1-i))(x - (-1+i))(x - (-1-i))$. The zeros are $1 \pm i$ $-1 \pm i$ (all imaginary).

CHAPTER PROJECT V

```
from matplotlib import pyplot as plt
from sympy import sqrt

def simplified_radical(x,y):
   x = round(x,0)
   y = round(y,0)
   num = int(x**2+y**2)

   # simplify the radical
   sq_rt = sqrt(num)

   # convert to a string
   sq_rt_str=str(sq_rt)
   # replace 'sqrt' with radical symbol
   sq_rt_str = sq_rt_str.replace('sqrt','\u221a')
   # remove '*'
   sq_rt_str = sq_rt_str.replace('*','')
   return sq_rt_str

# main program

# draw x and y axes
plt.plot([-10,10],[0,0],color='black')
plt.plot([0,0],[-15,15],color='black')

plt.xlabel("Real")
plt.ylabel("Imaginary")

z = complex(input('a+bj? '))
x1,y1 = z.real, z.imag

plt.plot([x1],[y1],'o')                    # plot the point

# Draw the line from origin to (x1,y1)
plt.plot([0,x1],[0,y1])

# calculate modulus (distance)

d = simplified_radical(x1,y1)
#d = abs(complex(x1,y1))
#d=round(d,2)

# write the complex number next to the point
```

```
plt.annotate(z,xy=(x1,y1))

# write the modulus next to the midpoint
xm = x1/2
ym = y1/2

plt.annotate(d,xy=(xm,ym))
plt.show()
```