SELECTED SOLUTIONS

CHAPTER 7

Exercise Set 7.1.1

1. plt.**plot(**[-10,10],[0,0],c='k') # draws x-axis
   plt.**plot(**[0, 0], [-50, 70],c='k') # draws y-axis

2. Write a function **draw_axes**(x_list,y_list) that will draw the x- and y-axes.

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

def draw_axes(x_list,y_list):
    x_min = min(x_list)
    x_max = max(x_list)
    y_min = min(y_list)
    y_max = max(y_list)
    plt.plot([x_min,x_max],[0,0],c='k')
    plt.plot([0,0],[y_min,y_max],c='k')
    return

x_list = range(-10,11)              # list of integers between -10 and 10
x = Symbol('x')
func1 = input('Function? ')
func1 = sympify(func1)              # convert from string to sympy expression

# assign y-values
y1_list = [func1.subs({x:x_value}) for x_value in x_list]

draw_axes(x_list,y1_list)
plt.plot(x_list,y1_list)
```

3. Replace

x_list = **range**(-10,11)

with

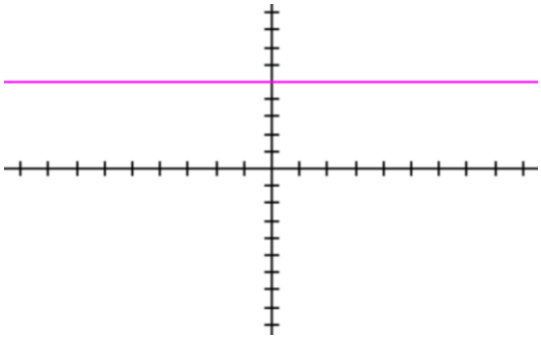
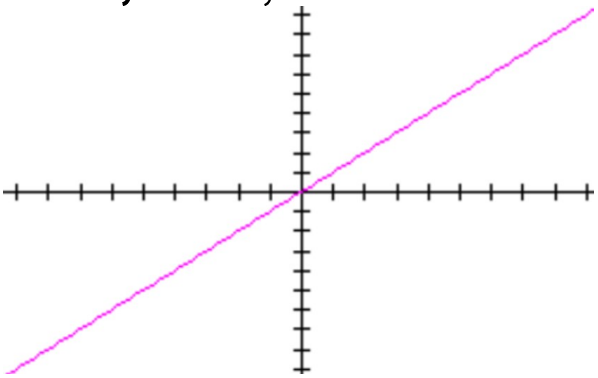
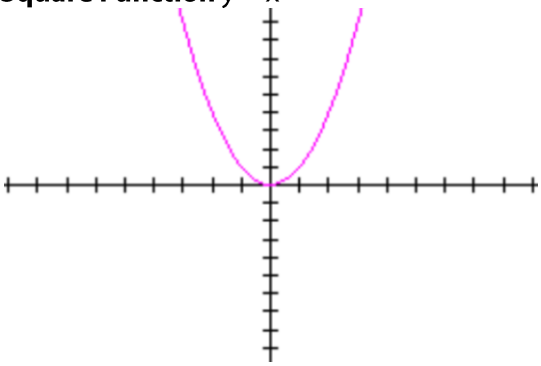
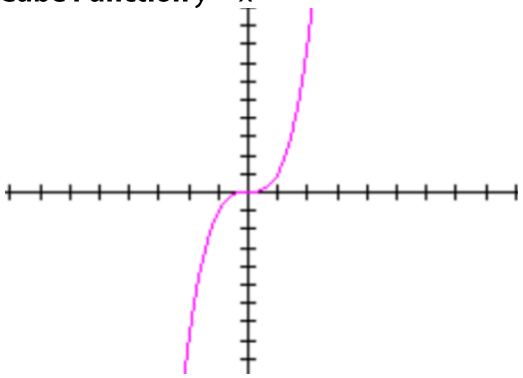
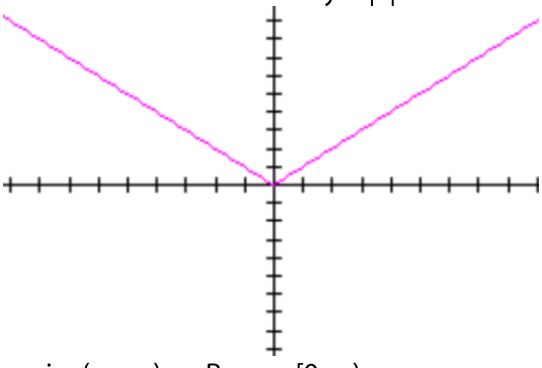
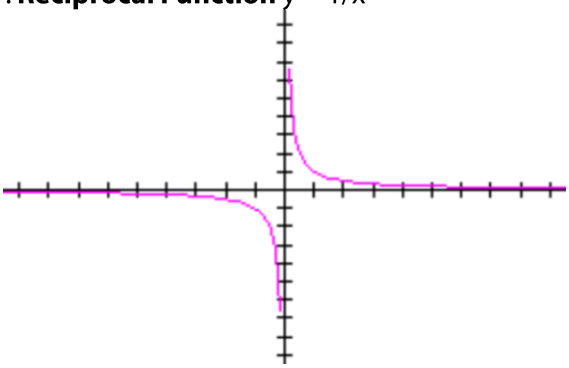
x_list = **range**(-20,21)

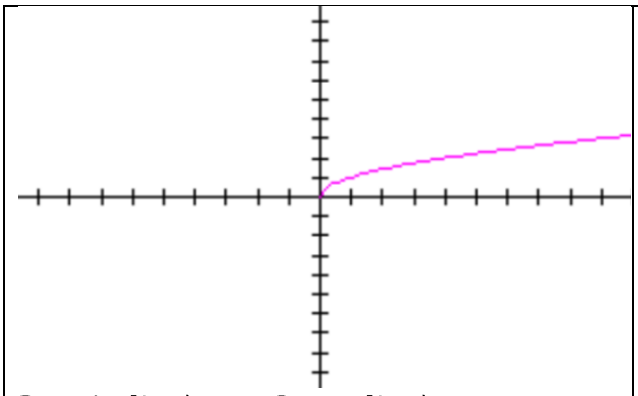
4. Append the following lines:

```
title = "The min is "+str(min(y1_list))+", the max is "+str(max(y1_list))
plt.title(title)
```

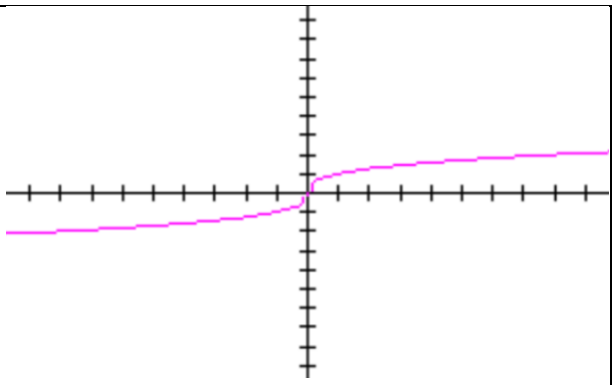
Exercise Set 7.1.2

1. Graph the following library functions and identify their domain and range.

A. **Constant Function** y = k  Domain: (-∞, ∞) Range: {k}	B. **Identity Function** y = x  Domain: (-∞, ∞) Range: (-∞, ∞)
C. **Square Function** y = x^2  Domain: (-∞, ∞) Range: [0, ∞)	D. **Cube Function** y = x^3  Domain: (-∞, ∞) Range: (-∞, ∞)
E. **Absolute Value Function** y = |x|  Domain: (-∞, ∞) Range: [0, ∞)	F. **Reciprocal Function** y = 1/x  Domain: (-∞,0) U (0,∞) Range: (-∞,0) U (0,∞)
G. **Square Root Function** y = $\sqrt{x}$	H. **Cube Root Function** y = $\sqrt[3]{x}$



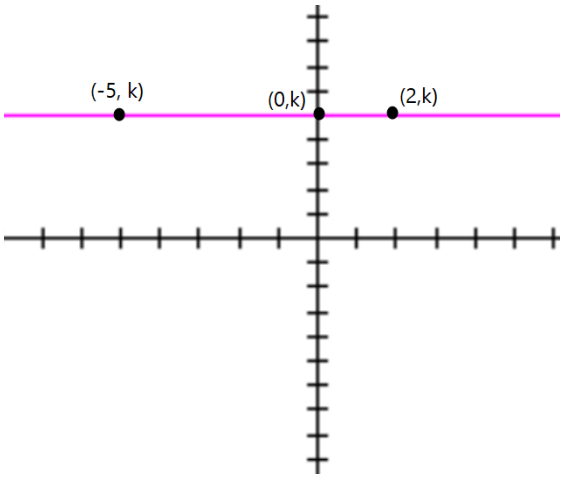
Domain: [0, ∞) Range: [0, ∞)



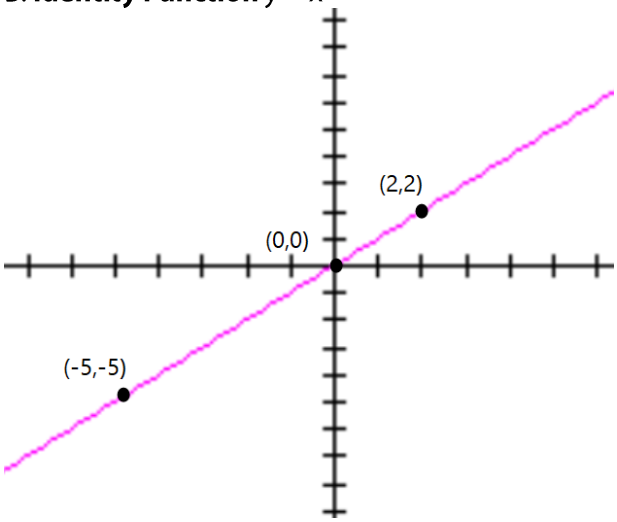
Domain: (-∞, ∞) Range: (-∞, ∞)

2. Graph the following library functions and plot the given points.

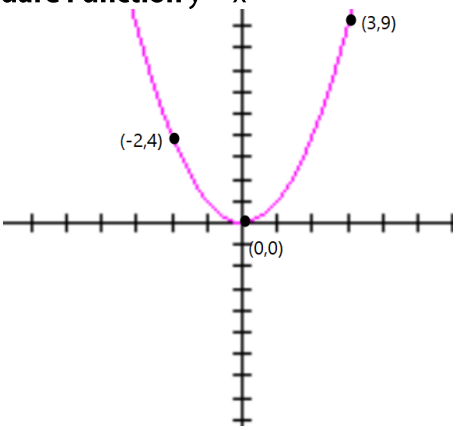
A. **Constant Function** y = k



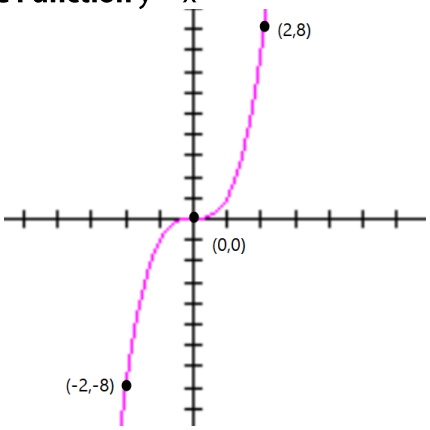
B. **Identity Function** y = x



C. **Square Function** $y = x^2$

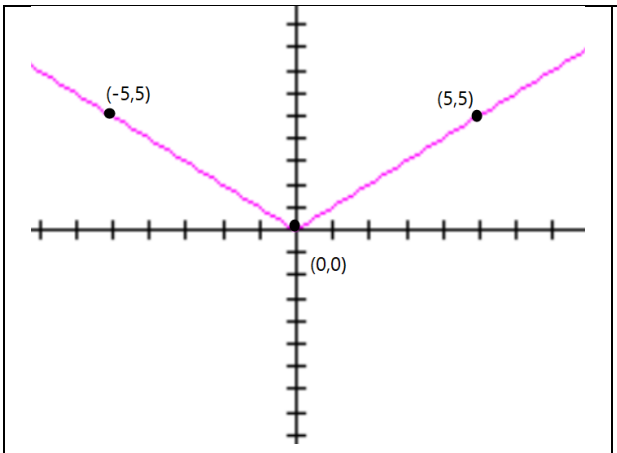


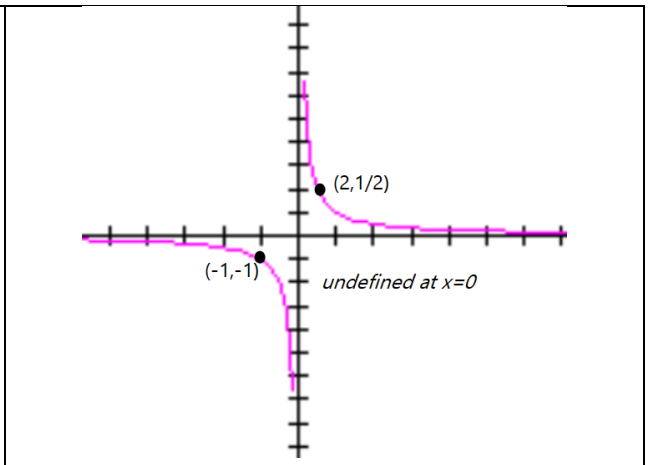
D. **Cube Function** $y = x^3$



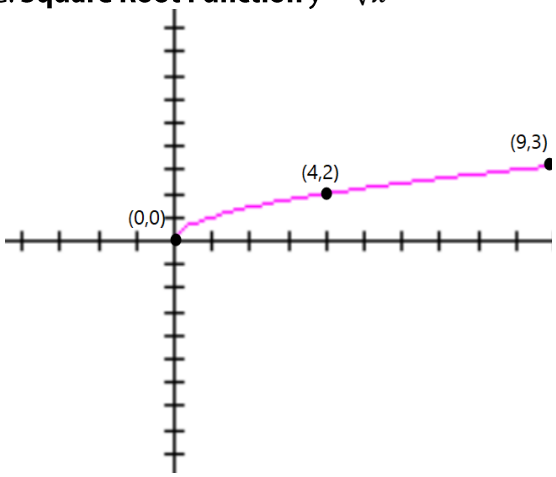
E. **Absolute Value Function** y = |x|

F. **Reciprocal Function** y = 1/x

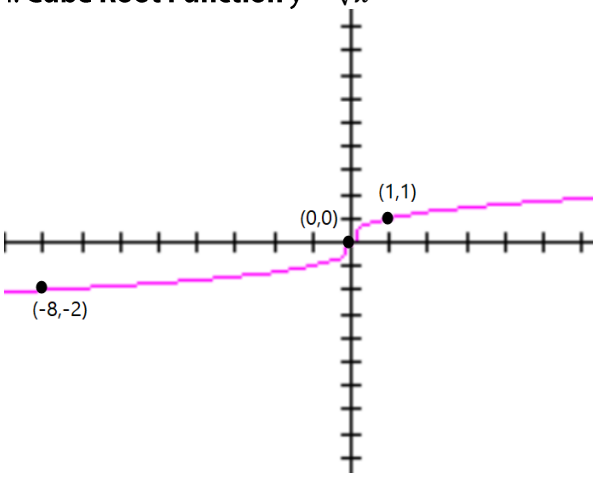




G. **Square Root Function** $y = \sqrt{x}$



H. **Cube Root Function** $y = \sqrt[3]{x}$



Exercise Set 7.1.3

1. Replace yellow highlighted code with:

```
y2_list=[func2.subs({x:x_value}) for x_value in x_list]
y3_list=[func3.subs({x:x_value}) for x_value in x_list]
```

2. Replace aqua highlighted code with:

```
plt.plot(x_list,y2_list)
plt.plot(x_list,y3_list)
```

3. `plt.title('Stretch and Compress')`

Exercise Set 7.1.4

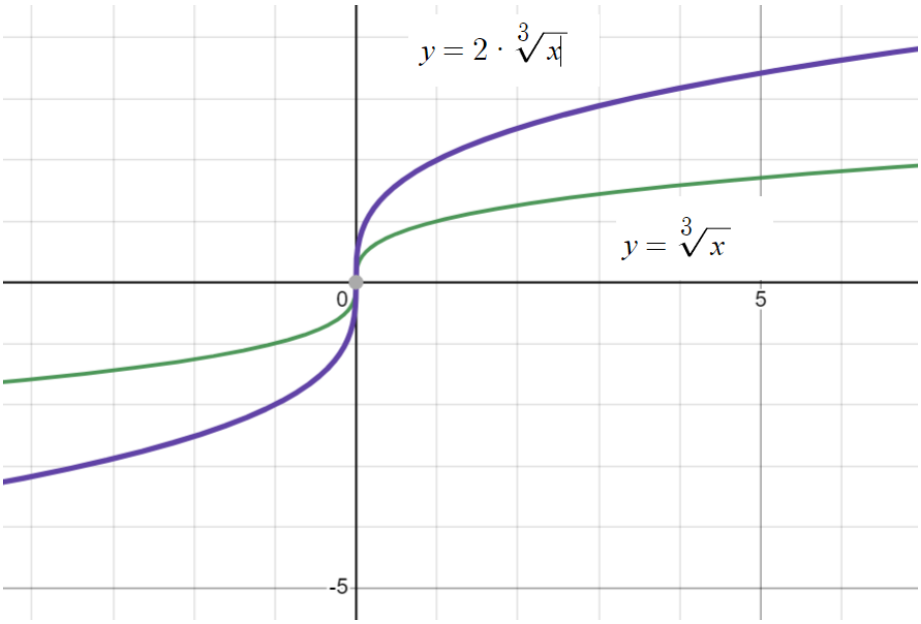
1. Modify program **stretch_compress** so that the user can input a function that is defined on [-10, 10]. Draw the graph of (1) the original function (2) the function stretched by a factor of 2 (3) the function compressed by a factor of ½. Include the legend.

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

x = Symbol('x')
func1 = sympify(input('Function? '))
func2 = expand(2*f)
func3 = expand(.5*f)

x_list = range(-10,11)
# assign y-values
y_list = [func1.subs({x:x_value}) for x_value in x_list]
y_list2 = [func2.subs({x:x_value}) for x_value in x_list]
y_list3 = [func3.subs({x:x_value}) for x_value in x_list]
plt.plot(x_list,y_list1, label=func1)
plt.plot(x_list,y_list2,label=func2)
plt.plot(x_list,y_list3,label=func3)
plt.legend()
plt.show()
```

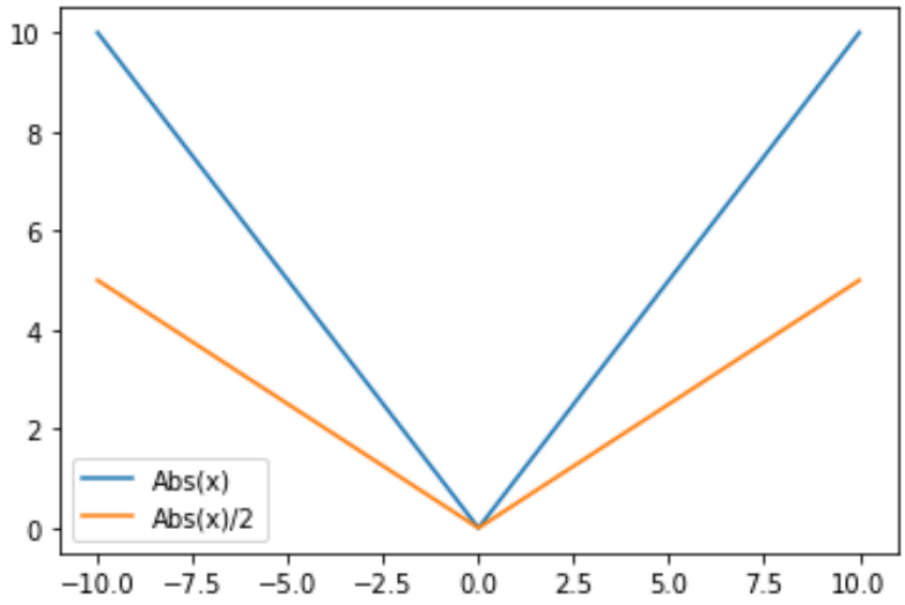
2. Modify the program so that the user can also specify the domain, rather than [-10, 10].

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

x = Symbol('x')
f = sympify(input('Function? '))
f2 = expand(2*f)
f3 = expand(.5*f)
domain = input('Domain? ')
xmin,xmax = [int(n) for n in domain.split(',')]
x_list = range(xmin,xmax)
# assign y-values
y_list = [f.subs({x:x_value}) for x_value in x_list]
y_list2 = [f2.subs({x:x_value}) for x_value in x_list]
y_list3 = [f3.subs({x:x_value}) for x_value in x_list]
plt.plot(x_list,y_list, label=f)
plt.plot(x_list,y_list2,label=f2)
plt.plot(x_list,y_list3,label=f3)
plt.legend()
plt.show()
```

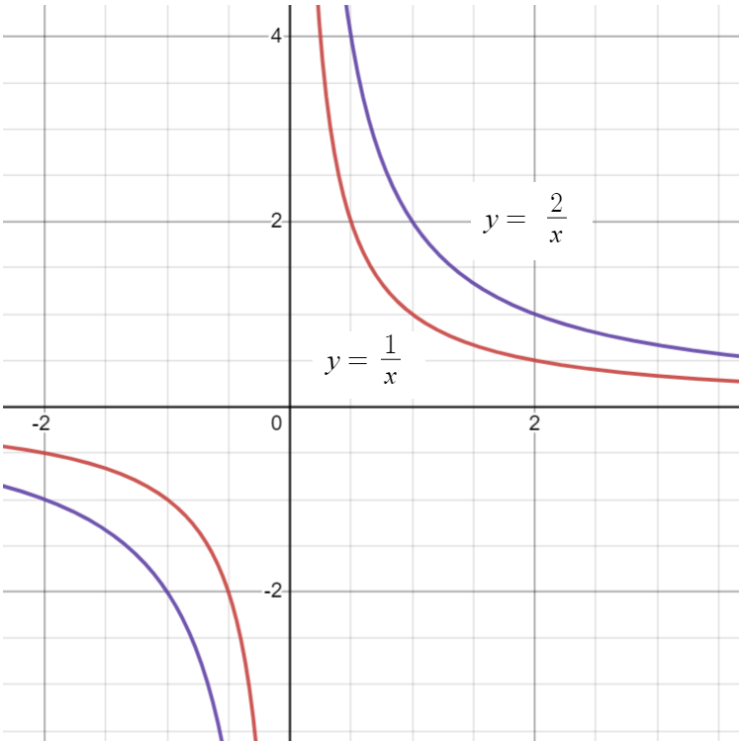
3.


$y = 2 \cdot \sqrt[3]{x}$
$y = \sqrt[3]{x}$
0
5
-5

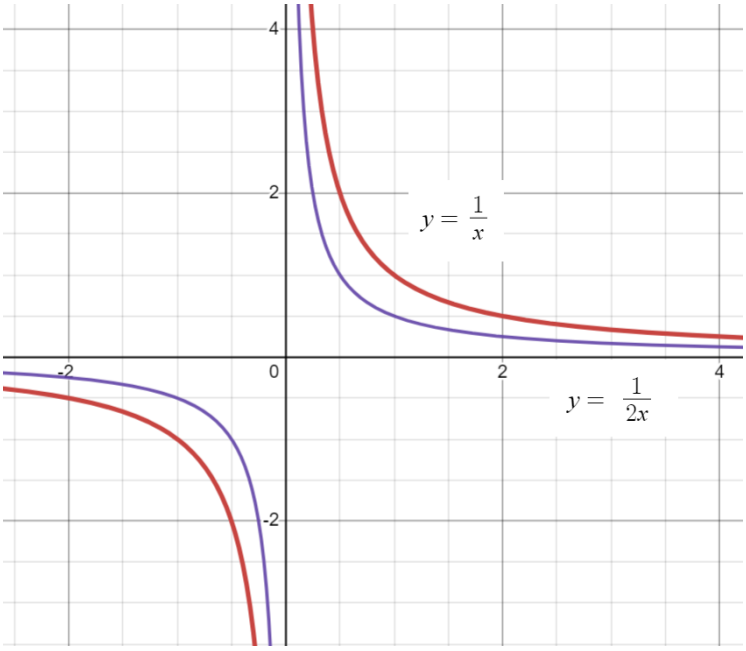
4.


10
8
6
4
2
0
Abs(x)
Abs(x)/2
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

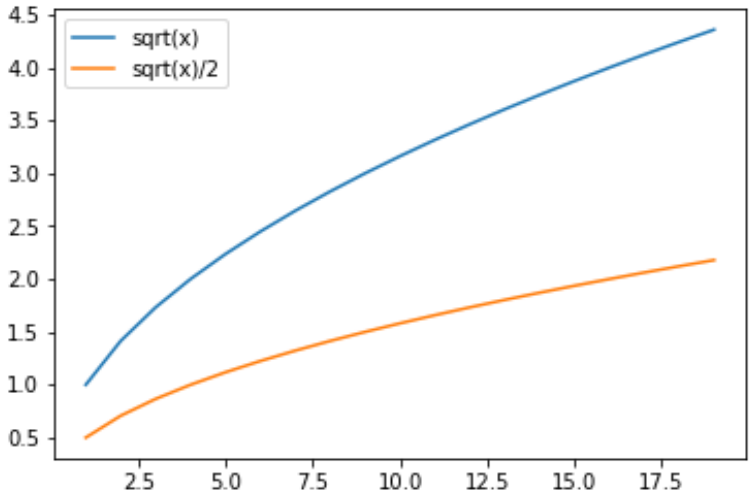
5.



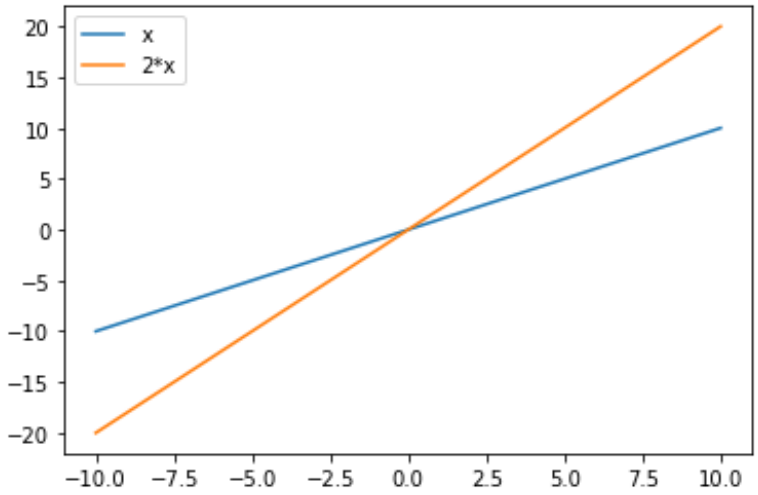
6.



7.



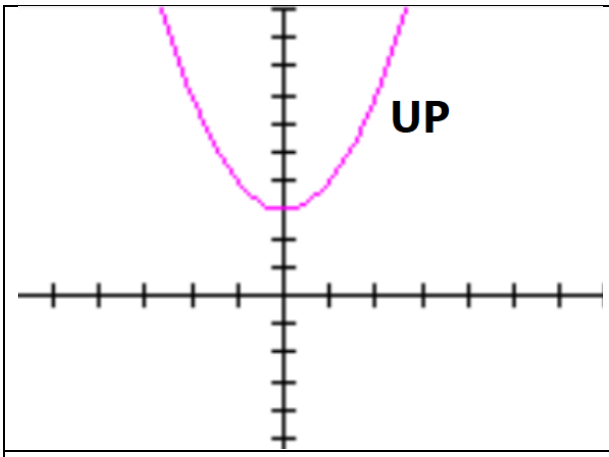
8.

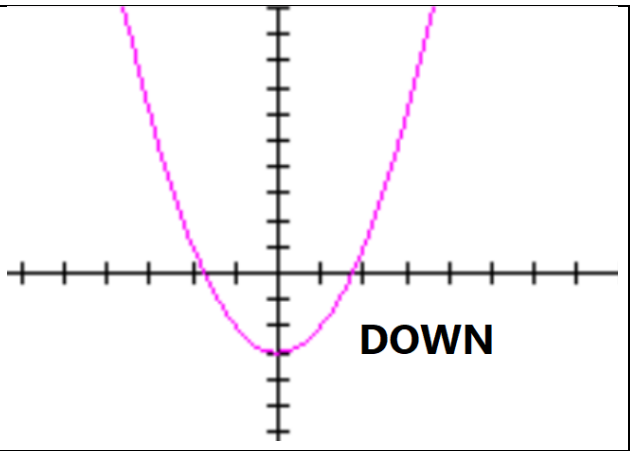


Exercise Set 7.2.1

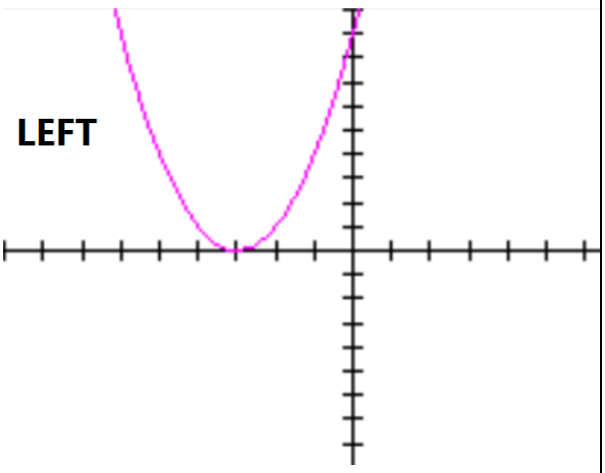
1. Sketch the graphs and state direction of movement.

A. $y = x^2 + k$	B. $y = x^2 - k$

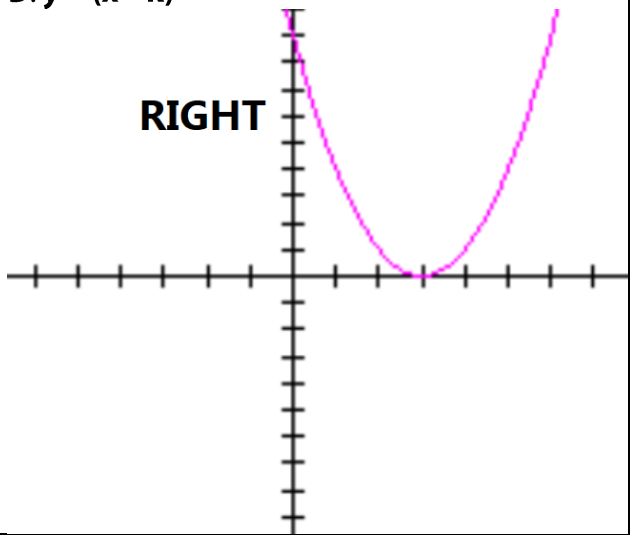




C. $y = (x + k)^2$

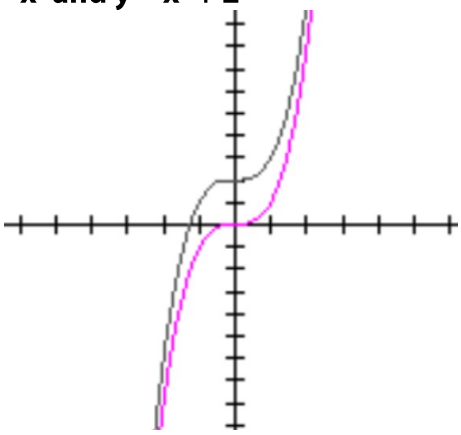


D. $y = (x - k)^2$

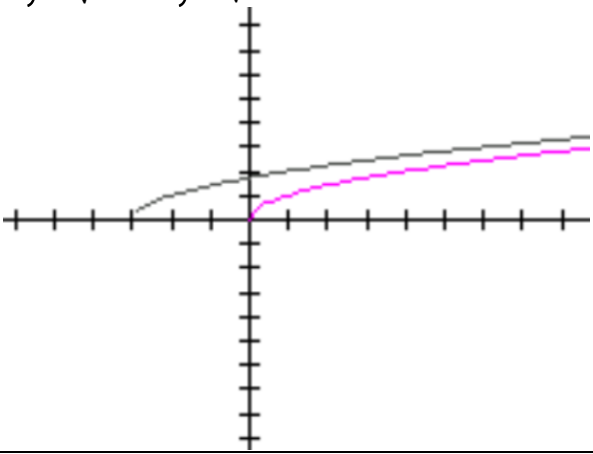


2. Sketch graph and identify y-intercept.

A. $y = x^3$ and $y = x^3 + 2$

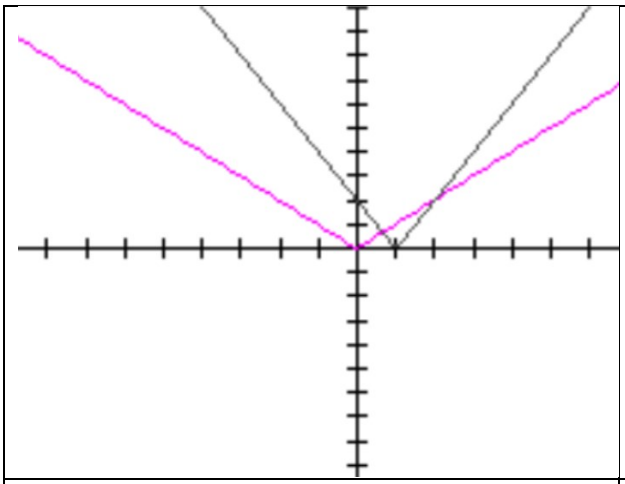


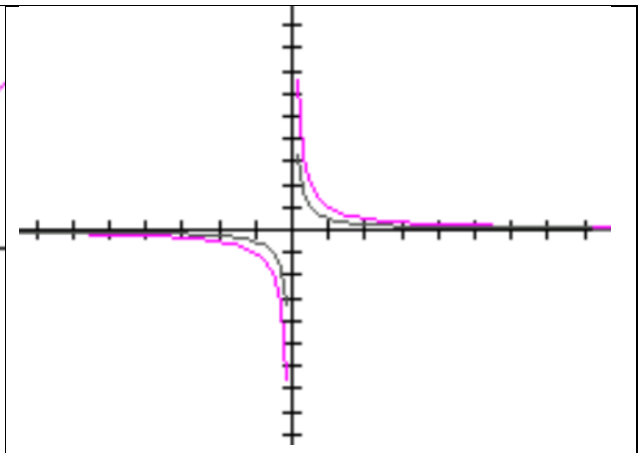
B. $y = \sqrt{x}$ and $y = \sqrt{x + 3}$



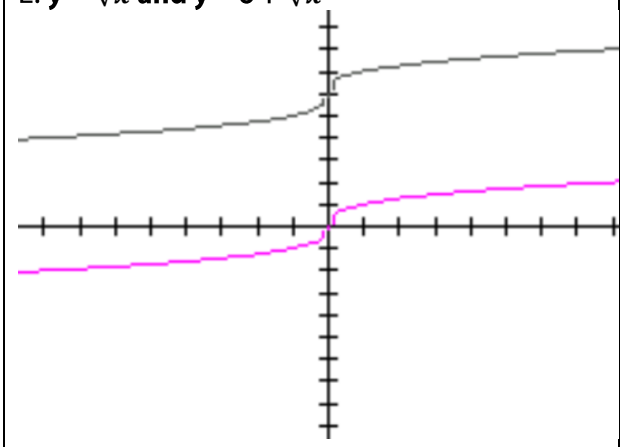
C. $y = |x|$ and $y = 2|x-1|$

D. $y = \frac{1}{x}$ and $y = \frac{1}{2x}$

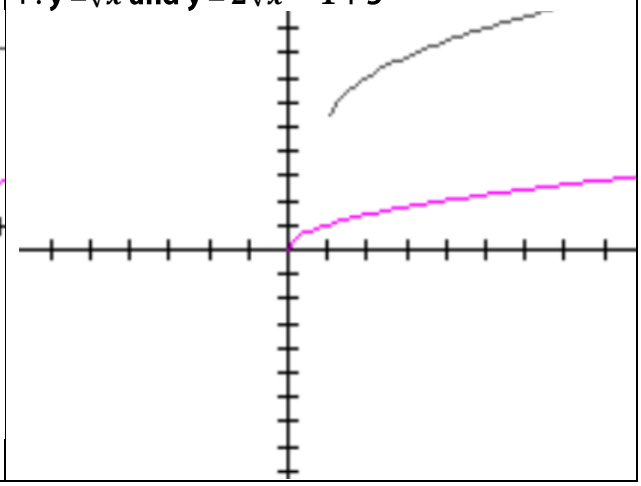




E. $y = \sqrt[3]{x}$ and $y = 6 + \sqrt[3]{x}$



F. $y = \sqrt{x}$ and $y = 2\sqrt{x-1} + 5$



Exercise Set 7.2.2

1.A. $y = \sqrt{x+1} - 3$

B. $y = \sqrt[3]{x-2}$

C. $y = \frac{5}{x+1}$

D. $y = 8x^3 - 1$

Exercise Set 7.2.3

1. Replace

    if (**not** y_value.**is_imaginary**):

with:

```
if (not y_value.is_imaginary) and (not y_value==S.ComplexInfinity):
```

or

```
if not (y_value.is_imaginary or y_value==S.ComplexInfinity):
```

2. Modify the program so that a 2nd function can be input and graphed.

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

def draw_axes(x_list,y_list):
    x_min = min(x_list)
    x_max = max(x_list)
    y_min = min(y_list)
    y_max = max(y_list)
    plt.plot([x_min,x_max],[0,0],c='k')
    plt.plot([0,0],[y_min,y_max],c='k')
    return

# main program
x_list = range(-10,11)

x = Symbol('x')
func1 = input("Function? ")
func1 = sympify(func1)

func2 = input("Function? ")
func2 = sympify(func2)

y1_list = [ ]
domain = [ ]
for x_value in x_list:
    y_value = func1.subs({x:x_value})
    if (not y_value.is_imaginary):
        domain.append(x_value)
        y1_list.append(y_value)

plt.plot(domain,y1_list)

y2_list = [ ]
domain2 = [ ]
for x_value in x_list:
    y_value = func2.subs({x:x_value})
    if (not y_value.is_imaginary):
        domain2.append(x_value)
        y2_list.append(y_value)
```

```
plt.plot(domain2,y2_list)

draw_axes(x_list,y1_list)

plt.show()
```

Exercise Set 7.2.4

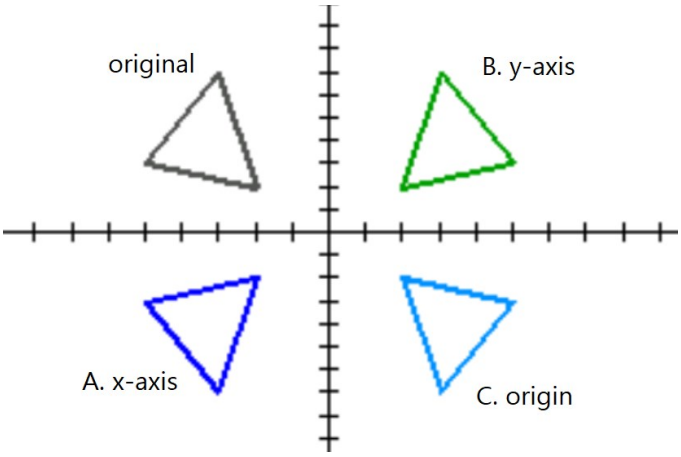
1. [-10.0, -9.9, -9.8, -9.7, … 9.7, 9.8, 9.9]
2. The only points plotted are those with x-values between -10 and 10.
3. A vertical shift does not affect the domain.
4. Set y_min to min(y_list) and y_max to max(y_list).

Exercise Set 7.3.1

1. Fill in yellow highlighted code to output the reflected point through the y-axis and origin:

```
print('Reflection through y-axis:', -x,y)
print('Reflection through origin:', -x,-y)
```

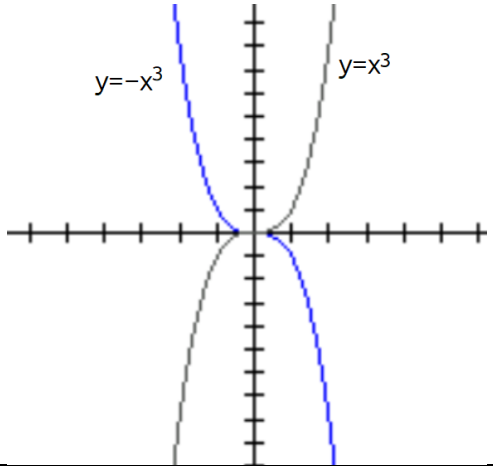
2. Given triangle with vertices (-2,2), (-5,3) and (-3,7), draw its reflections.



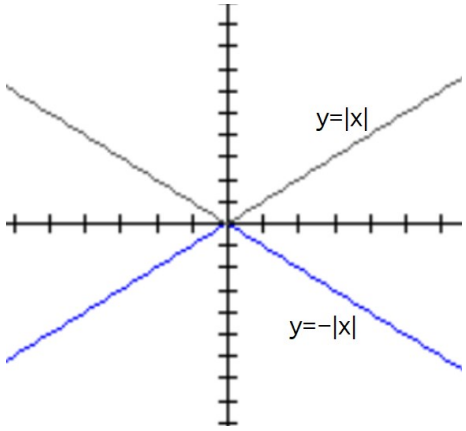
3.

Exercise Set 7.3.2

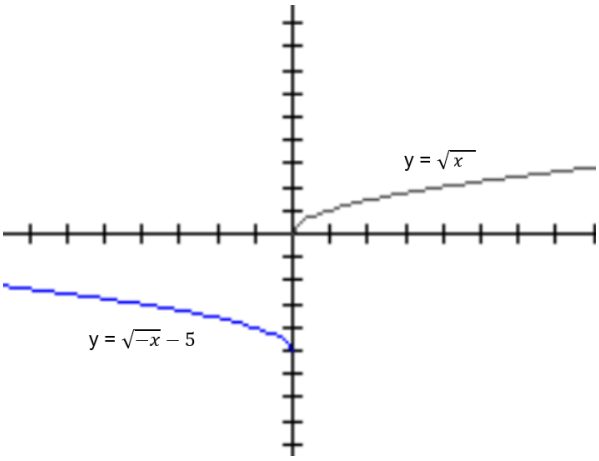
1.



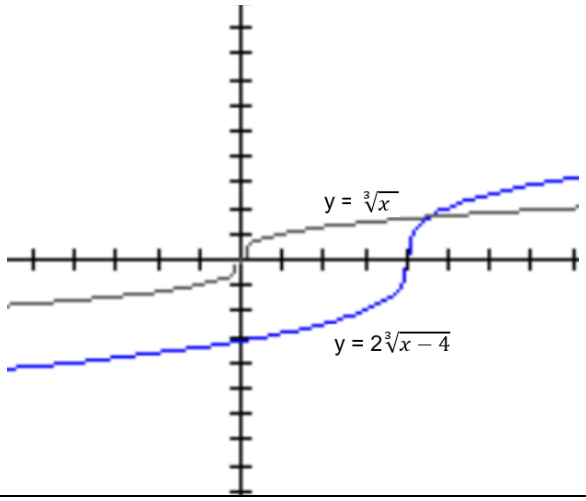
2.



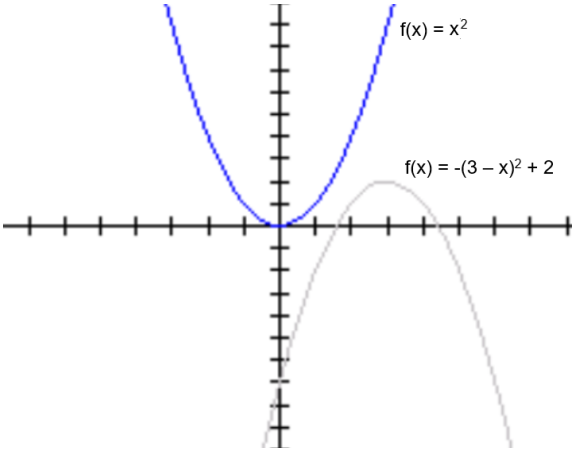
3.



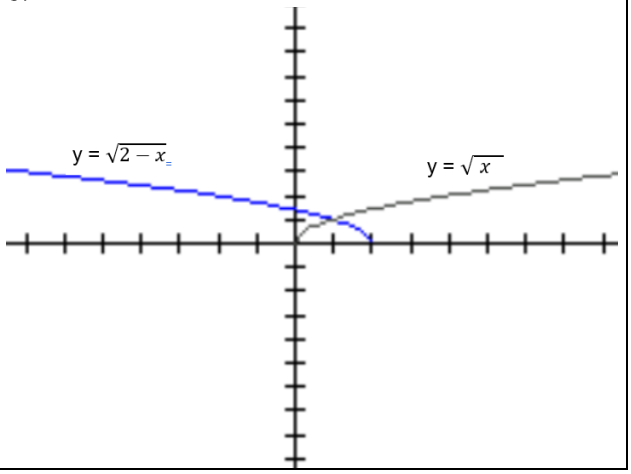
4.



5.



6.



7. A. y = -|2x| or y = -2|x|

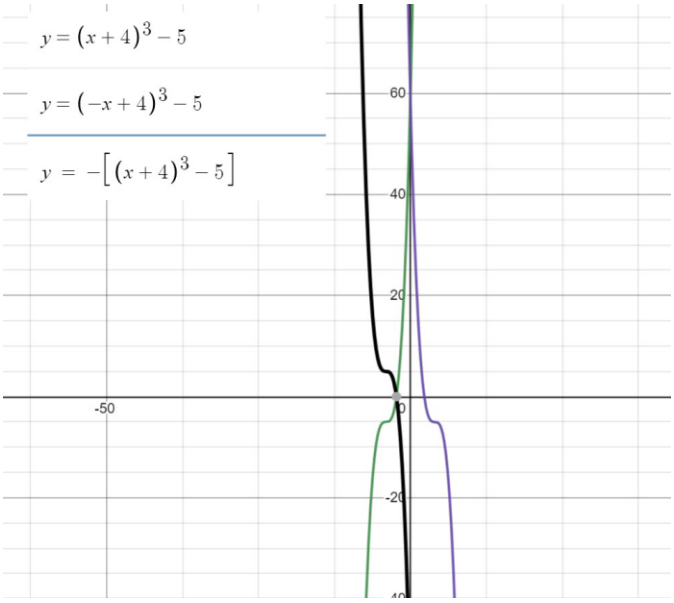
B. $y = -2\sqrt[3]{x}$

C. $y = \frac{1}{2}(x-1)^2$

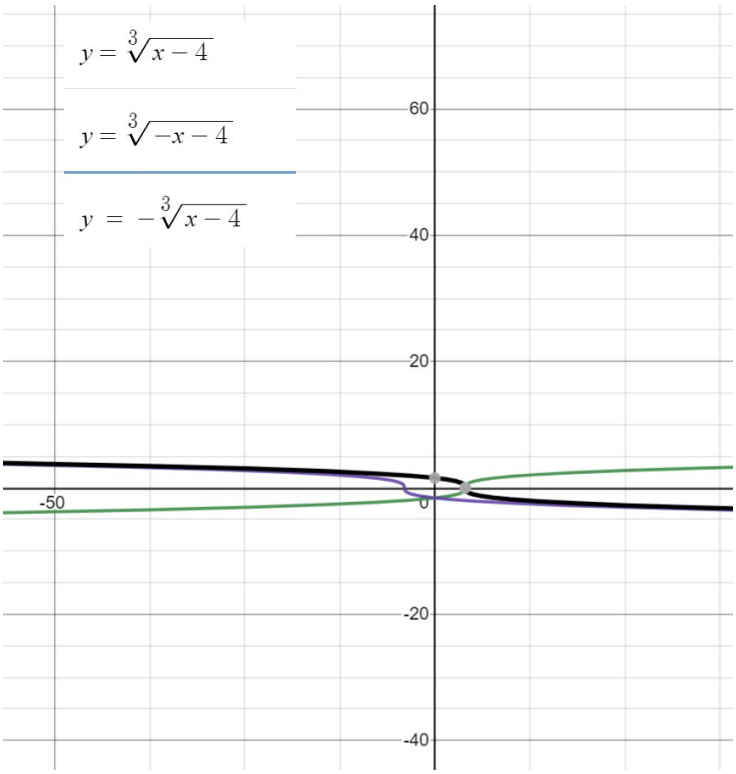
D. $y = -\frac{10}{x-5}$

Exercise Set 7.3.3

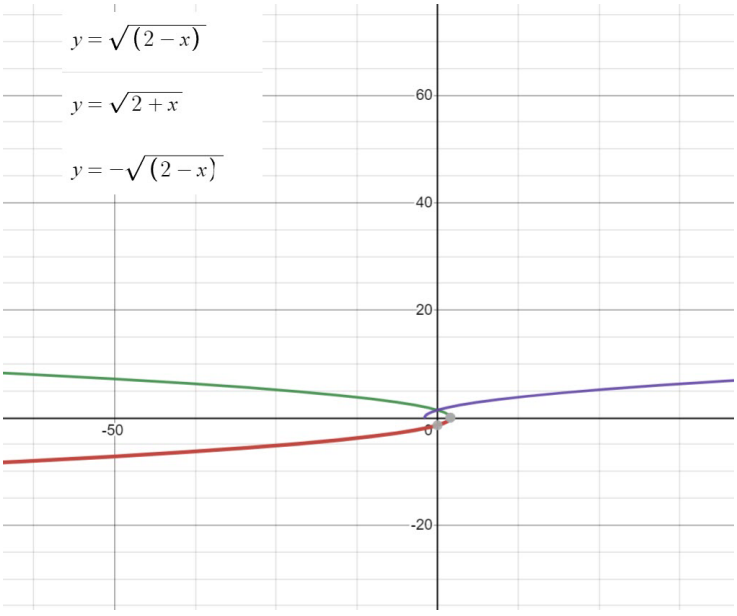
1.



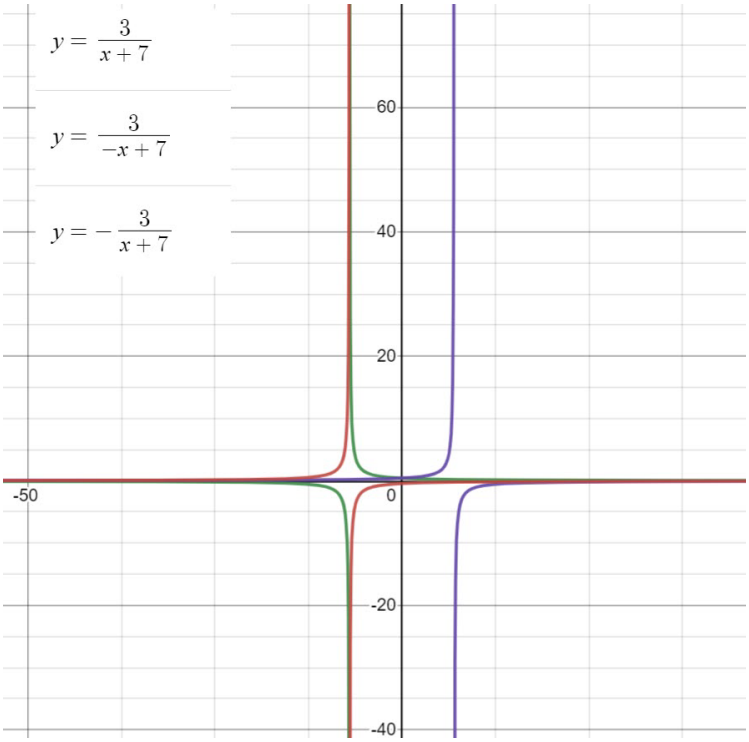
2.


$y = \sqrt[3]{x-4}$
$y = \sqrt[3]{-x-4}$
$y = -\sqrt[3]{x-4}$
60
40
20
-50
0
-20
-40

3.


$y = \sqrt{(2-x)}$
$y = \sqrt{2+x}$
$y = -\sqrt{(2-x)}$
60
40
20
-50
0
-20

4.



Exercise Set 7.4.1

1. None

2. Origin

3. y-axis

4. origin

5. y-axis

6. origin

7. origin

8. y-axis

9. y-axis

10. origin

11. No, a graph with x-axis symmetry can not be considered a function because by its very definition it fails the vertical line test.

Exercise Set 7.4.2

1. Even

2. Odd

3. Neither

4. Neither

5. Not every graph with y-axis symmetry can be considered a function. A circle has y-axis symmetry but is not a function.

6. Not every graph with origin symmetry can be considered a function. A circle has origin symmetry but is not a function.

Exercise Set 7.4.3

1. Fill in the yellow highlighted code to output 'f(-x)=' followed by pretty print of f(-x).

```
f2 = sympify(f2)
print('f(-x)=')
pprint(f2)
```

2. Modify the program to determine if the input function is odd, even, or neither by comparing f(x) to f(-x).

```
from sympy import *

x = Symbol('x')
f = input('Function? ')
f2 = f.replace('x','(-x)')

f = sympify(f)
print('f(x)=')
pprint(f)

f2 = sympify(f2)
print('f(-x)=')
pprint(f2)

if f == f2:
   print("even function")
elif f == -f2:
   print("odd function")
else:
   print("neither even nor odd")
```

Exercise Set 7.5.1

1. Increasing: (-6, -5), (0,2); Decreasing: (-4,0), (2, 5); Constant: (-10, -6), (5, 9);
Abs max: 8; local max: 4; abs min: 0

2. Increasing: (-6, -4), (-2, 0), (3, 5); Decreasing: (-8, -6), (0, 3), (5,7); Constant: (-4, -2);
Abs. max: 5; local max: 4; abs min: -4; local min: 3

3. Increasing: (-∞, 0.6), (1, 1.3), (1.7, 2), (2.3, 2.7); Decreasing: (1.5, 1), (1.3, 1.7), (2, 2.3), (2.7, 2.8);
Constant: nowhere; Abs max: 1.5; local max: 0.7, 0.5, 0.4; Abs min: none; local min: -1, -0.6, -0.4

4. Increasing: (-∞, ∞)

5. Increasing: (0, ∞); Decreasing: (-∞, 0)

6. Increasing: (0, ∞)

7. Increasing: (2, ∞); Decreasing: (-∞, 2)

8. A. The graph of a function can have a local maximum or absolute minimum at an endpoint.
False

B. The graph of a function can have an absolute maximum or absolute minimum at an endpoint.
True

C. A point on a graph can be both an absolute and local maximum.
False

D. The graph of a function can have more than one absolute maximum or absolute minimum.
True

E. The graph of a function can be neither increasing nor decreasing.
True (if it is a horizontal line)

F. Increasing, decreasing, and constant intervals are open intervals of x-values.
True

G. A function is neither increasing nor decreasing at a maximum or minimum point.
True

H. The graph of every function has both increasing and decreasing intervals.
False. For example, $y = \sqrt{x}$ is increasing over its entire domain of [0, ∞).

9. No the program would not be 100% accurate. As a counterexample, let's say "X^2" was input as the function and {-10, 10} as the interval; and the program randomly generated A=-5 and B=6, then it would observe that $25 < 36$ and deduce that the function was increasing. However $f(x) = x^2$ is not increasing over (-10, 10).

Exercise Set 7.5.2

1. A. The program calculates an absolute max of -0.25.

B. The actual max is 0 at x = 0.5.

2. Modify the program so that the user can enter the domain rather than using [-10, 10].

```
from sympy import Symbol,sympify

x = Symbol('x')
f = input('Function? ')
domain = input('Domain? ')
xmin,xmax = domain.split(',')
a = int(xmin)
b = int(xmax)+1
f = sympify(f)
x_list = range(a,b)
y_list = [f.subs({x:x_value}) for x_value in x_list]
y_min = min(y_list)
y_max = max(y_list)
print('Absolute minimum:',y_min)
print('Absolute maximum:',y_max)
```

3. Yes, this is called the Extreme Value Theorem.

Exercise Set 7.5.3

1. Fill in the yellow highlighted code to calculate and output the ordered pair of the absolute maximum.

```
y_max = max(y_list)
x_max = get_x(y_max)
print('Absolute maximum:',x_max, ',', y_max)
```

2. A. Absolute minimum: 10 , -2971
Absolute maximum: -10 , 2949

B. Absolute minimum: -1 , -13
Absolute maximum: 10 , 10987

3. Modify the program so that it graphs the given function and labels the absolute maximum and absolute minimum on the graph with the ordered pairs.

```
from sympy import Symbol,sympify
from matplotlib import pyplot as plt

def get_x(y):
    global x_list,y_list
    k = y_list.index(y)
    return x_list[k]
```

```
x = Symbol('x')
f = input('Function? ')

f = sympify(f)
x_list = range(-10,11)
y_list = [f.subs({x:x_value}) for x_value in x_list]
y_min = min(y_list)
x_min = get_x(y_min)
y_max = max(y_list)
x_max = get_x(y_max)

plt.plot( x_list, y_list)
str1 = '('+str(x_min)+','+str(y_min)+')'
str2 = '('+str(x_max)+','+str(y_max)+')'
plt.text(x_min,y_min,str1)
plt.text(x_max,y_max,str2)
```

Exercise Set 7.6.1

1. A. 4

   B. 4

   C. 0

   D. 9

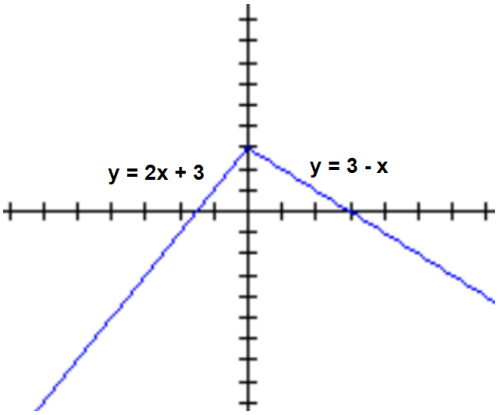
   E. 9

2. $$f(x) = \begin{cases} -x+2 & if\ -5 \le x < -2 \\ 4 & if\ -2 \le x < 0 \\ -x^2+4 & if\ 0 \le x < 2 \\ x-2 & if\ x \ge 2 \end{cases}$$

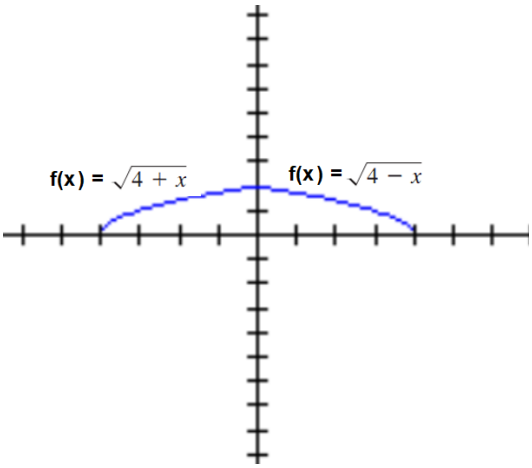
3. $$f(x) = \begin{cases} x & if\ x \ge 0 \\ -x & if\ x < 0 \end{cases}$$

4. $$f(x) = \begin{cases} -2 & if\ -2 \le x < -1 \\ -1 & if\ -1 \le x < 0 \\ 0 & if\ 0 \le x < 1 \\ 1 & if\ 1 \le x < 2 \end{cases}$$

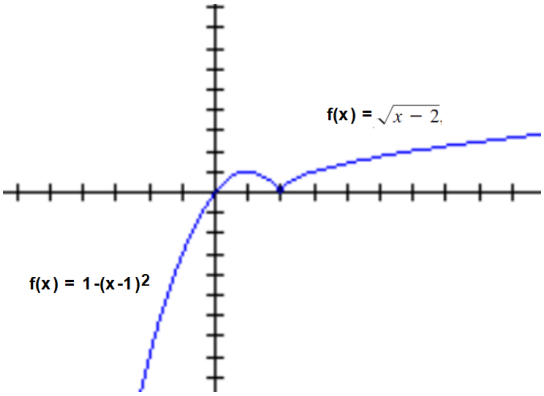
5. A.



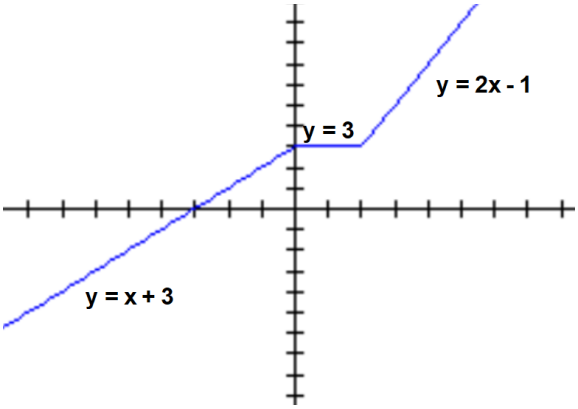
B.



C.



D.



Exercise Set 7.6.2

1. Append the following lines:

```
y2_list=[piece2.subs({x:x_value}) for x_value in domain2]
plt.plot(domain2,y2_list,label=piece2)

y3_list=[piece3.subs({x:x_value}) for x_value in domain3]
plt.plot(domain3,y3_list,label=piece3)
```

2. Modify the program to draw the given piecewise-defined function.

```
from matplotlib import pyplot as plt
from sympy import Symbol,sympify

x=Symbol('x')

piece1 = sympify('4')
piece2 = sympify('-(x+4)**2+8')
piece3 = sympify('x**2')
piece4 = sympify('(-4/3)*x+(20/3)')

domain1 = [i/10 for i in range(-100,-60)]
domain2 = [i/10 for i in range(-60,-20)]
domain3 = [i/10 for i in range(-20,20)]
domain4 = [i/10 for i in range(20,50)]

fig = plt.figure()
ax = plt.subplot()
plt.axis([-10,10,-20,20])

y1_list = [ ]
y2_list = [ ]
y3_list = [ ]
y4_list = [ ]
```

```
y1_list=[piece1.subs({x:x_value}) for x_value in domain1]
plt.plot(domain1,y1_list,label=piece1)
y2_list=[piece2.subs({x:x_value}) for x_value in domain2]
plt.plot(domain2,y2_list,label=piece2)
y3_list=[piece3.subs({x:x_value}) for x_value in domain3]
plt.plot(domain3,y3_list,label=piece3)

y4_list=[piece4.subs({x:x_value}) for x_value in domain4]
plt.plot(domain4,y4_list,label=piece4)

ax.legend()
plt.show()
```

3. Rewrite program so that user can input 3 functions and 3 domains.

```
from matplotlib import pyplot as plt
from sympy import Symbol,sympify

x=Symbol('x')

p1 = input('piece 1? ')
p2 = input('piece 2? ')
p3 = input('piece 3? ')

d1 = input('domain 1? ')
d2 = input('domain 2? ')
d3 = input('domain 3? ')

a1,b1 = d1.split(',')
a2,b2 = d2.split(',')
a3,b3 = d3.split(',')

a1,b1 = int(a1),int(b1)
a2,b2 = int(a2),int(b2)
a3,b3 = int(a3),int(b3)

piece1 = sympify(p1)
piece2 = sympify(p2)
piece3 = sympify(p3)

domain1 = [i for i in range(a1,b1)]
domain2 = [i for i in range(a2,b2)]
domain3 = [i for i in range(a3,b3)]


fig = plt.figure()
```

```
ax = plt.subplot()
plt.axis([-10,10,-20,20])

y1_list = [ ]
y2_list = [ ]
y3_list = [ ]


y1_list=[piece1.subs({x:x_value}) for x_value in domain1]
plt.plot(domain1,y1_list,label=piece1)
y2_list=[piece2.subs({x:x_value}) for x_value in domain2]
plt.plot(domain2,y2_list,label=piece2)
y3_list=[piece3.subs({x:x_value}) for x_value in domain3]
plt.plot(domain3,y3_list,label=piece3)

ax.legend()
plt.show()
```

Exercise Set 7.6.3

1. Replace the np.**piecewise**() function with:

```
np.piecewise(x, [x<-2,x>=-2 and x<2,x>=2 and x<=5,x>5],[x*(x+2)**2,-x**2+4,-(x-2),x-5])
```

A. f(-2) = 0

B. f(0) = 4

C. f(2) = 0

D. f(5) = -3

E. f(6) = 1

2. It returns float type.

Exercise Set 7.6.4

1. Append the following lines to the program:

```
# x,y axes
plt.plot([-10,10],[0,0],color='k')
plt.plot([0,0],[-20,20],color='k')
# annotations
plt.text(-10,-7,'y=x+3')
```

```
plt.text(0,3,'y=3')
plt.text(2,3,'y=2*x-1')
```

2. Replace the np.**piecewise**() function with:

```
np.piecewise(x,[x>=-10 and x<0, x>=0 and x<1,x>=1 and x<3,x>=3],[x,-(x-1)**2+1,1,2*(x-3)**2+1])
```