SELECTED SOLUTIONS

CHAPTER 11

Exercise Set 11.1.1

1. 3

2. 1/3

3. 100,000

4. ½

5. $\log_5 125 = 3$

6. $\log_{81} 9 = ½$

7. $\log_{11}(1/121) = -2$

8. $\log_{216} 6 = 1/3$

9. $7^2 = x$

10. $10^{x+8} = 3$

11. $5^x = 125$

12. undefined

13. 0

14. undefined, the base cannot be 1

15. $(-2, \infty)$

Exercise Set 11.1.2

1. log base 27 of 8.999999999999998 = (2/3) (Exact answer is $\log_{27} 9 = 2/3$)

2. The output is **log base (-3) of 81 = 4** which is not entirely accurate because you cannot perform a logarithm with a negative base. Insert the following blue code and indent the lines following the **else:**

```
 :
 :
else:
    a=base_exp[0]
    b = base_exp[1]
```

```
    if float(a)<=0:
      print("The base must be a positive number.")
    else:
      y = eval(ex_str2)
      print("log base",str(a),"of",str(y),"=",str(b))
```

3. No, the **if float(a)<=0:** line returns an error because the program is trying to convert a fraction to a float. Modify the following by inserting the following blue code:

```
:
:
if len(base_exp)!=2:
   print("Your exponential statement must include **")
else:
   a=base_exp[0]
   b = base_exp[1]
   if a.find('/')>0:              # a is a fraction
      f = a.split('/')            # split the fraction at the '/'
      n = float(f[0])             # convert numerator to float
      d = float(f[1])             # convert denominator to float
      a = Fraction(n/d)
:
:
```

4. Insert the following blue code:

```
 :
 :
 if float(a)<=0:
       print("The base must be positive")
    elif float(a)==1:
       print("The base cannot be one")
    else:
 :
 :
```

5. Import **Fraction** from **fractions** and insert the following blue code at the end of the program:

```
   else:
      y = eval(ex_str2)
      y = Fraction(y).limit_denominator()
      print("log base",str(a),"of",str(y),"=",str(b))
```

Exercise Set 11.1.3

1. $\log_4 0.25 = -1$
2. $\log_9 27.0 = 3/2$
3. $\log_5 0.04 = -2$

4. $\log_8 3.9999999999999996 = 2/3$ (exact answer is $\log_8 4 = 2/3$)
5. $\log_7 343 = 3$
6. No, the program will only work if the base is a single digit.
7. Replace float number with fraction.
   Insert:
   ```
   from fractions import Fraction
   ```
   at beginning of program, and insert:
   ```
   y = Fraction(y).limit_denominator()
   ```
   after:
   ```
   y = eval(ex_str)
   ```

8. Write a program that converts a logarithmic expression in the form log(x,b) (representing logbx), calculates its value, and outputs a corresponding exponential equation in the form b**y = x.

```
from math import log

f = input("logarithmic function log(x,b)? ")

start = f.index('(')
end = f.index(')')
argument = f[start+1:end]
parameters = argument.split(',')
x = parameters[0]
b = parameters[1]
y = eval(f)
print(b,"**",y,"=",x)
```

Exercise Set 11.1.4

1. 0

2. 3

3. 1

4. y

5. 1 – 0 = 1

6. $\log_b 1 = 0$ because in exponential form, $b^0 = 1$ for $b > 0$.

7. $\log_b b = 1$ because in exponential form, $b^1 = b$ for $b > 0$.

8. $\log_b 0$ is undefined because there is no number x such that $b^x = 0$.

Exercise Set 11.2.1

1. $f^{-1}(x) = \log_5(x+3)$

2. $f^{-1}(x) = log_5\left(-\frac{x}{2}\right)$

3. $f^{-1}(x) = \left(\frac{1}{2}\right)^x - 5$

4. $f^{-1}(x) = 10^{-x}$ or $f^{-1}(x) = \left(\frac{1}{10}\right)^x$

5. $f^{-1}(x) = \frac{1}{3} log_3\left(\frac{x+1}{2}\right)$

6. $f^{-1}(x) = \frac{1-e^{1-x}}{2}$

7. $y = b^x$ and $y = \log_b x$ are both one-to-one. This can be observed by noting that their graphs pass the horizontal line test.

8. Write a program that asks user to input domain and range of a function f. Output the domain and range of its inverse f^{-1}.

```
domf = input("domain of f? ")
ranf = input("range of f? ")
print("domain of f inverse is",ranf)
print("range of f inverse is",domf)
```

Exercise Set 11.2.2

1. What is the output if the user inputs log(7*x-49) ?
   x > 7

2. If the user inputs 'log(4*x+2)', identify the values and types of the following variables:
   start: 3, integer
   end: 9, integer
   argument: 7*x+1, str
   undefs: [-1/7], list
   undef: -1/7, sympy rational (this varies depending on nature of solution)

3. What does f.**index**('(') do?
   identifies the position of '(' in the string variable f.

4. What does **solve**(argument) do?
   It solves the equation argument = 0.

5. What is the output if the user inputs log(49-7*x)? Is this the correct solution? Why or why not?
   The output is x > 7.
   This is incorrect, the domain is actually x < 7 which is the solution to 49-7x > 0.

6. This program is accurate if the argument of the function is a linear expression in the form ax+b, in which a > 0. Is it accurate if the argument is a linear expression in the form ax+b in which a < 0? If not, fix the program.
   It is not accurate, as we saw in #5 above. One way to fix the program is to choose a test_point on the right of the undefined point. If f(test_point) > 0, then the domain is everything to the right of the undefined point. Otherwise, the domain is everything to the left of the undefined point.

```
from sympy import *
x = Symbol('x')
f = input('logarithmic funtion? ')
start = f.index('(')
end = f.index(')')
argument = sympify(f[start+1:end])
undefs=solve(argument)
undef = undefs[0]
test_point = undef+1
f_test = argument.subs({x:test_point})
if f_test > 0:
   print('x>',undef)
else:
   print('x<',undef)
```

7. What is the output if the user inputs log(x**2 – 4)?
   x< -2
   This is not accurate. The actual solution to $x^2 - 4 > 0$ is $(-\infty, -2) \cup (2, \infty)$.

8. Is it accurate if the argument is a quadratic, such as log(x**2 – 4) ?
   No, the program would need to perform a quadratic inequality to solve $ax^2 + bx + c > 0$.

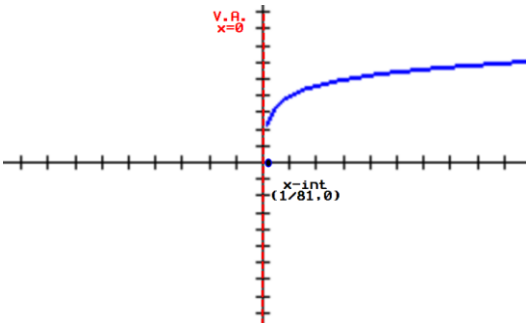
Exercise Set 11.3.1

1. Domain: (0,∞); Range: (-∞, ∞)

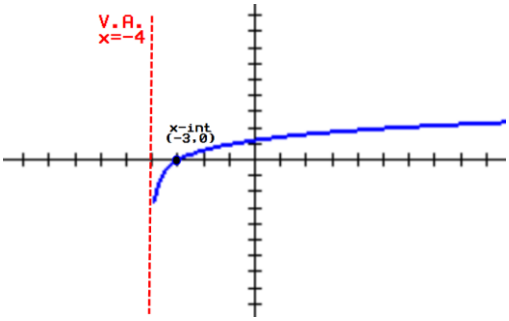
2. V.A.: x=0

3. x-int: (1,0); no y-int

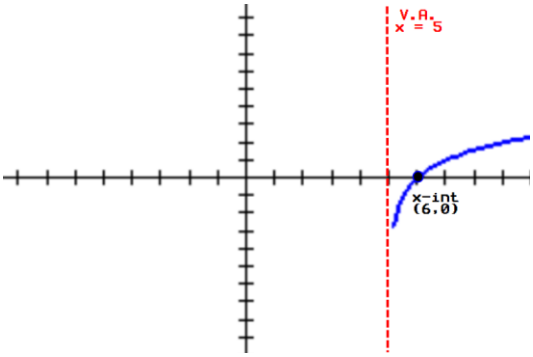
4. $f(x) = \log_3 x + 4$



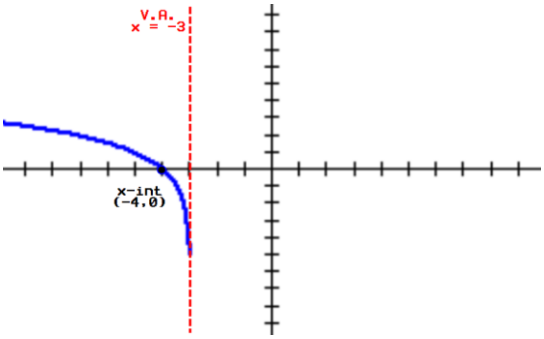
5. $f(x) = \log_3(x+4)$



6. $f(x) = -\log_{1/2}(x-5)$



7. $f(x) = \log_2(-x-3)$



8. Modify program **log_domain** so that rather than identifying the domain of a log function, it identifies the vertical asymptote.

```
from sympy import solve
f = input('logarithmic function? ')
start = f.index('(')
end = f.index(')')
argument = f[start+1:end]
undefs=solve(argument)
VA = undefs[0]
print('vertical asymptote is x=',VA)
```

Exercise Set 11.4.1

1. $e^{\ln x} = x$

   $\ln e^{x} = x$

   $\ln 1 = 0$

   $\ln 0$ is undefined

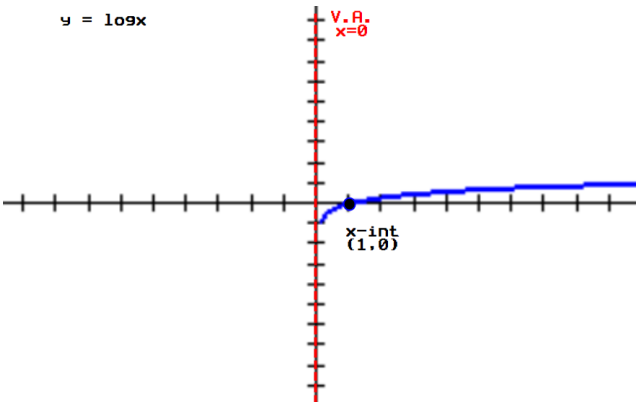
2. $1 + 1 = 2$

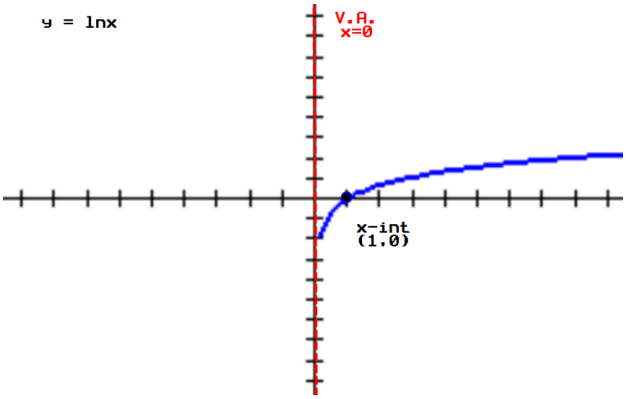
3. 3

4. 3

5. -2

6. Graphs of $\log x$ and $\ln x$





Exercise Set 11.4.2

1. 6.0
2. 15.0
3. math domain error
4. 2.0
5. 15.0
6. 36.0
7. -2.0
8. The value of n
9. Write a program that finds the x-intercept of a logarithmic function.

```
from sympy import solve
f_str = input('logarithmic funtion? ')
x_int=solve(f_str)
print('The x-intercept of', f_str, 'is (', x_int[0], ',0)')
```

10.

```
from sympy import *
from math import log
# array of subscripts
subscripts=['\N{SUBSCRIPT ZERO}','\N{SUBSCRIPT ONE}',
        '\N{SUBSCRIPT TWO}','\N{SUBSCRIPT THREE}',
        '\N{SUBSCRIPT FOUR}','\N{SUBSCRIPT FIVE}','\N{SUBSCRIPT SIX}',
        '\N{SUBSCRIPT SEVEN}','\N{SUBSCRIPT EIGHT}',
        '\N{SUBSCRIPT NINE}']
f = input('logarithmic function? ')

start = f.index('(')
end = f.index(')')
argument = f[start+1:end]
parameters = argument.split(',')
x = parameters[0]
b = parameters[1]
if b != 'e' and len(b)==1:
   base = subscripts[int(b)]
   y = eval(f)
   print('log'+base,x,'=',y)
elif b=='e':
   expr = 'log('+x+')'
   y=eval(expr)
   print('ln',x,'=',y)
else:
   print('I don\'t know how to do that')
```

Exercise Set 11.5.1

1. $1 + \log x$

2. $1 + 2\log_4 x$

3. $\frac{1}{2} \ln 3 + \frac{1}{2} \ln x - \ln 7$

4. $\ln x + \ln y - \ln z$

5. $\frac{1}{3}\log_3(x-2)$

6. $5\log_b x + 2\log_b z - 3\log_b y$

7. $\ln(-b+\sqrt{b^2-4ac}) - (\ln 2 + \ln x)$

8. $\log \frac{7}{x}$

9. $\log_5 \frac{\sqrt{7}}{x^2}$

10. $\ln \frac{x^3y^2}{z^4}$

11. $\log_4 4x^3$

12. $\ln \frac{x^9}{x^6} = \ln x^3$

13. $\ln \sqrt{x^2+y^2}$

14. $2t + u$

15. $3u - 4t$

16. $\frac{2}{3}u + \frac{2}{3}t$

17. $\frac{4}{3}r + \frac{2}{3}t$

18. $f^{-1}(x) = \frac{1}{6} \cdot 10^{2x}$

19. Let $x = \log_b M$ and $y = \log_b N$
    Then $b^x = M$ and $b^y = N$
    So $\log_b \frac{M}{N} = \log_b \frac{b^x}{b^y} = \log_b b^{x-y} = x - y = \log_b M - \log_b N$

20. Let $x = \log_b M$ then $b^x = M$

So $\log_b M^N = \log_b (M + M + \ldots\ldots + M)$ (N terms)

$= \log_b M + \log_b M + \ldots\ldots + \log_b M$ (N terms)

$= N \cdot \log_b M$

Exercise Set 11.5.2

1. $\frac{ln81}{lnb}$

2. $\frac{lnx}{ln2}$

3. False

4. False

5. True

6. True

7. False

8. False

9. True

10. False

11. False

12. Write a program that asks the user to input a log expression in the form log(x,b) where x and b are constants. Calculate the value of log(x,b) using the change-of-base formula log(x)/log(b). Assume the base is a single integer digit.

```
from sympy import *
from math import log
# array of subscripts
subscripts=['\N{SUBSCRIPT ZERO}','\N{SUBSCRIPT ONE}',
        '\N{SUBSCRIPT TWO}','\N{SUBSCRIPT THREE}',
        '\N{SUBSCRIPT FOUR}','\N{SUBSCRIPT FIVE}','\N{SUBSCRIPT SIX}',
        '\N{SUBSCRIPT SEVEN}','\N{SUBSCRIPT EIGHT}',
        '\N{SUBSCRIPT NINE}']

f = input('logarithmic function? ')

start = f.index('(')
```

```
end = f.index(')')
argument = f[start+1:end]
parameters = argument.split(',')
x = parameters[0]
b = parameters[1]
if b != 'e' and len(b)==1:
   base = subscripts[int(b)]
   expr = 'log('+x+')/log('+b+')'
   y = eval(expr)
   print('log'+base,x,'=',expr,'=',y)
```

Exercise Set 11.5.3

1. math.**log**(8,3)
1.892789260714372

2. log(8,3)
log(8)/log(3)
$\frac{ln8}{ln3}$

3. **simplify**(log(x*y)
log(x·y)
ln(x·y)

4. **expand_log**(log(x*y)
log(x) + log(y)
lnx + lny

5. **simplify**(log(4,a)+log(5,a))
log(20)/log(a)
$\frac{\log(20)}{\log(a)}$ which is equivalent to $\log_a 20$

6. **simplify**(log(x,a)-log(y,a)+log(z,a))
log(x*z/y)/log(a)
$\frac{log\left(\frac{x \cdot z}{y}\right)}{\log(a)}$ which is equivalent to $log_a\left(\frac{x \cdot z}{y}\right)$

7. **simplify**(a*log(x)+log(y)-log(z))
log(x**a*y/z)

$log\left(\frac{x^a \cdot y}{z}\right)$ which is equivalent to $ln\left(\frac{x^a \cdot y}{z}\right)$

8. **simplify**(log(x*y**2))
log(x*y**2)
ln(x·y^2)

9. **expand_log**(log(x*y**2))
log(x) + 2·log(y)
lnx + 2·lny

10. **expand_log**(log((x**2*y)/z**3,a))
(2*log(x) + log(y) - 3*log(z))/log(a)
$\frac{2 \cdot \log(x) + \log(y) - 3 \cdot \log(z)}{\log(a)}$ which is equivalent to 2·$\log_a$x + $\log_a$y – 3·$\log_a$z.

11. **logcombine**(3***log**(x))

log(x**3)

ln(x^3)

12. **logcombine**(log(x) + 3***log**(y))

log(x*y**3)

ln(x·y^3)

13. Not possible using **simplify** or **expand_log**.

14. Not possible using **simplify** or **expand_log**.

15. **simplify**(log(x**a))

16. **simplify**(**log**(**sqrt**(a*b)))

17. **logcombine**(3***log**(a)+4***log**(b),**force**=True)

18. >>>**print**(sympy.**ln**(e))
1.00000000000000

>>> **print**(math.**ln**(e))
module 'math' has no attribute 'ln'

>>> **print**(math.**log**(e))
1.0

Exercise Set 11.5.4

1. >>>**expand_log**(**log**(x*y/z))
   log(x) + log(y) – log(z)

2. >>>**simplify**(**log**(x,y)) (or **expand_log**)
   log(x)/log(y)

3. >>>**simplify**(**log**(**sqrt**(x*y))) (or **expand_log**)
   log(x)/2 + log(y)/2

4. >>>**simplify**(**log**(x,z) + 2***log**(y,z)) (or **logcombine**)
   log(x*y**2)/log(z)

5. >>>**simplify**(**ln**(x**2)+**ln**(**sqrt**(x))) (or **logcombine**)
   5*log(x)/2

Exercise Set 11.6.1

1. $x = 7/2$

2. $x = 37/63$

3. $x = 4$

4. $x = 12$

5. $x = 4/3$

6. $x = 1 - \log_3 2 \approx .3691$

7. $x = -\ln 4 \approx -1.3863$

8. $x = \frac{e^2}{5}$

9. $x = \frac{e^3+5}{6}$

10. ± ½

11. **solve**(log(4*x + 11,5) - 2)

12. **solve**(log(x+5,2)-log(2*x-1,2)-5)

13. Not possible to do with a single **solve**.

14. **solve**(log(x,6)+log(x-9,6)-2)

15. **solve**(log(x,10)-log(2*x-3,10)-log(2,10))

16. **solve**(3**(1-x)-2) (returns -log(2)/log(3) + 1)

17. Not possible to do with a single **solve**.

18. **solve**(log(5*x)-2)

19. **solve**(log(6*x-5)-3)

20. **solve**((1/4)*log(e)-x**2)

21. Write a program to solve an exponential or logarithmic equation.

```
from sympy import *
equation = input("Enter exponential or logarithmic equation? ")
left,right = equation.split('=')
equation2 = left+"-"+right
answers = solve(equation2)
print(answers)
```

Exercise Set 11.6.2

1. $x = \ln 5 \approx 1.6094$

2. $x = 0, x = \log_3 5 \approx 1.450$

3. $x = 2$

Exercise Set 11.7.1

1. A. $P(t) = 10000(\frac{1}{2})^{t/40}$ or $P(t) \approx P_0 e^{-.017329t}$

B. 28,284 bacteria

C. 92.88 minutes

2. A. 5832

B. 3.9 days

3. A. $k = \frac{\ln(\frac{1}{2})}{5730} \approx -0.000121$

B. 2949 years

4. 16899 years

5. 26.6 days

6. A. 9.999999467 g

B. 9.999994668 g

7. 9.9 yr

8. Write a program that will calculate amount of time t to reach a given amount P(t).

```
from math import log
h = float(input('Half-life? '))
a0 = float(input('Initial amount? '))
at = float(input('Ending Amount? '))
k = log(0.5)/h
t = (1/k)*log(at/a0)
print('Time to reach',at,'is',round(t,0))
```

CHAPTER PROJECT VI

Insert this function into program **convert_to_log**:

```
def form_log(a,y):
    # returns expression in the form log_a y

    # array of subscripts
    subscripts=['\u2080','\u2081', '\u2082', '\u2083', '\u2084', '\u2085', '\u2086', '\u2087',
          '\u2088', '\u2089']
```

```
    statement ="log"
    a=list(str(a))                    # forms list of digits of a
    a = [int(d) for d in a]           # convert string digits to int
    for d in a:                       # for each digit in a
        statement += subscripts[d]  # concatenate the corresponding subscript string
    statement += str(y)

    return statement                  # returns log_a y
```

Then replace the final **print** statement in the main program to:

```
result = form_log(a,y)
result += " = "+str(b)
print(result)
```

CHAPTER PROJECT VIII

```
from math import log10

words = ['one','ten','hundred','thousand','ten-thousand','hundred-thousand','million','ten-million',
'hundred-million','billion','ten-billion','hundred-billion','trillion']

number = int(input("Multiple of 10? "))

# check that the input number is a multiple of 10
if number%10 != 0:
    print(number,"is not a multiple of 10")
else:

    exponent = int(log10(number))

 # check that the input number is less than one trillion
    if exponent>12:
        print(number,"is too high for me.")
    else:
        print(words[exponent])
```

CHAPTER PROJECT IX

```
## negative_exponent.py
## input: a = e**(-x) where a is a positive number and x is a variable
## output: solution to the equation in terms of ln, and decimal equivalent
```

```
import re
from fractions import Fraction
from math import log

expr = input("Exponential expression a=e**(-x)? ")
pattern1 = "^\d+=e\*\*\(\-.+\)$"

if re.match(pattern1,expr):
   # isolate the constant on left side of equation
   sides=expr.split('=')            # split into two sides of the equation
   a = float(sides[0])

   # isolate the x
   ex =sides[1].replace("e**(-","")
   ex = ex.replace(")","")


   recip_a = 1/a                    # float representation of 1/a
   recip_a_frac = Fraction(1/a).limit_denominator(1000)

   print(ex,'= ln(',recip_a_frac,') =',round(log(recip_a),4))
else:
   print("input must be in form a=e**(-x)")
```