Lab 1: Review of Basic Commands

Before we begin applying Python to Calculus, let's take a week to review (or learn, if you did not take 151 here) the basic commands of the symbolic package of Python, sympy.

EXAMPLE:

An open-top box is made by cutting out equal square corners of an 9 x 12 inch sheet of cardboard and folding up the flaps (see similar picture in 1.1 #63)

1) Write and expand a formula V(x) for the volume of the box as a function of the length x of the squares.

2) Find the volume when the squares are 2 1/8 inches to a side.

3) Find the length of the sides of the square required to get a volume of 50 cubic inches.

4) Plot a graph of V(x) in an appropriate practical domain.

Of course, nothing in Python can be done here without YOU doing the first step-writing the function V(x)! After a little thought, you should see that V(x) = x *(9-2x)* (12 - 2x).

Once you have your function, you are ready to use Python. The symbolic package is a "library" of commands, so we will start all of our labs with the next set of commands which allow us to import all of the commands in sympy (i.e., "check out all the library commands")

In [1]:
```
from sympy import *
from sympy.plotting import (plot, plot_parametric)
```

1. Defining Symbolic Expressions

Now we are ready to define our symbolic variable x and start solving the problem. The key to #1 is the word "expand"

In [2]:
```
x=symbols('x',positive=True)
# This allows Python to treat the letter x as a variable.  Since x is a len
gth, we force it to be positive
V=x*(9-2*x)*(12-2*x)
print(V.expand())
```

```
4*x**3 - 42*x**2 + 108*x
```

We mentioned these last week in the Course Overview, but two important things to notice in the above commands and output. First, as expected, the "expand" command expands our volume function, but the syntax is VERY different-and very common to Python! In most cases, when you want to perform a command on a variable, the correct Python syntax is

variable.command

instead of "command(variable)". The second thing to notice is the Python output, specifically how exponents are used. Instead of printing 4 * x^3 (as done on a calculator, for example), Python used two stars for the exponent. This is because the ^ is a logical operator in Python. Inputs are the same: use x ** 2 instead of x^2.

2. Substituting Into Symbolic Expressions

For question 2), we need to SUBSTITUTE the value 2 1/8 = 2.125 into x. A quick check of help documentation shows that the **subs** command does the trick.

In [3]:
```
Vsub=V.subs(x,2.125)
print('The volume when x=2.125 is',Vsub)
```

```
The volume when x=2.125 is 78.2265625000000
```

Again, notice the syntax: *Variable.Command(Arguments)*. Also notice that we can include explanatory text in our print statement. By "can", I mean "should always". :)

3. Solving Equations Symbolically (Exact)

For question 3) we need to solve V(x) = 50. There is a type "Equation" in Python, but it is generally easiest to move everything to one side and use the expression, in this case, solve V - 50 = 0.

In [4]:
```
Vsoln=solve(V-50,x)
print('The values of x which make V=50 are',Vsoln)
```

```
The values of x which make V=50 are [7/2 + (-1/2 - sqrt(3)*I/2)*(15/8 + sqr
t(493)*I/4)**(1/3) + 13/(4*(-1/2 - sqrt(3)*I/2)*(15/8 + sqrt(493)*I/4)**(1/
3)), 7/2 + 13/(4*(-1/2 + sqrt(3)*I/2)*(15/8 + sqrt(493)*I/4)**(1/3)) + (-1/
2 + sqrt(3)*I/2)*(15/8 + sqrt(493)*I/4)**(1/3), 7/2 + 13/(4*(15/8 + sqrt(49
3)*I/4)**(1/3)) + (15/8 + sqrt(493)*I/4)**(1/3)]
```

Notice that Python gives bizarre looking solutions which appear to be complex (they all have "I" in them). There are two possible ways to resolve this issue. The first is to convert the answers to floating-point decimals using the command **evalf**. HOWEVER, notice the square brackets around our answers. Recall this is a type "list", and you cannot convert a list to floating point! You can, however, convert each element of the list in one command by using a Python tool called **list comprehension**.

In [5]:
```
VsolnAPPROX=[i.evalf() for i in Vsoln]
# The literal translation of the above command is "VsolnAPPROX is a list fo
und by applying the evalf command
# (i.e., converting it to a floating point decimal) to each element in the
 list 'Vsoln'."
print('The values of x which make V=50 are',VsolnAPPROX)
```

```
The values of x which make V=50 are [3.10926620932673 + 0.e-22*I, 0.5912574
6933059 + 0.e-20*I, 6.79947632134268 - 0.e-20*I]
```

Notice that each solution DOES have an imaginary portion, but a careful observation shows that it is infintessimally small: "e-22" is scientific notation, or "times 10^(-22)", so they can be ignored.

A second way to obtain the solutions is to actually proceed with problem #4 and use the graph to solve the equation numerically.

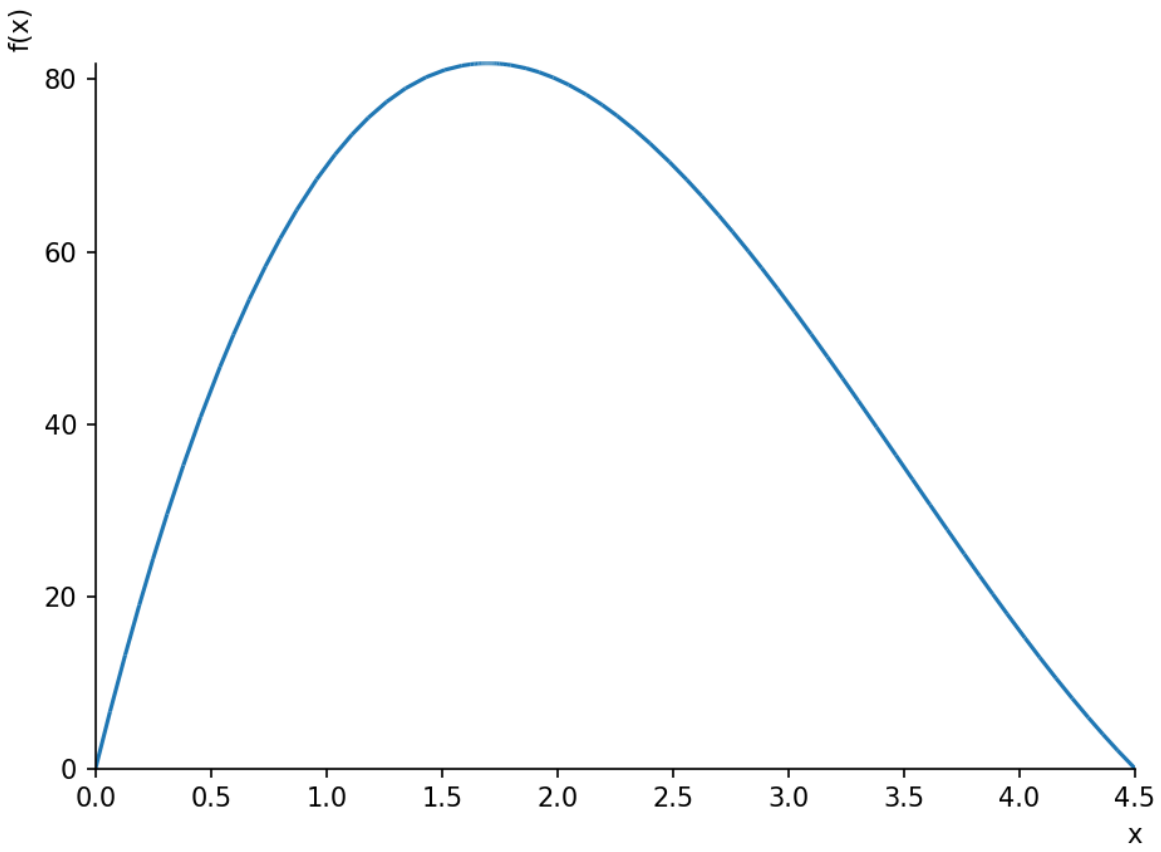
4. Plotting Symbolic Expressions

In Jupyter, before each graph, you need to enter the following command:

In [6]:
```
matplotlib notebook
```

This allows Python to produce the graph in the Jupyter notebook (and not included on a previous graph). For a practical domain, we notice each of the terms we multiplied in our Volume must be positive. So 0<=x<=4.5.

In [7]:
```
plot(V,(x,0,4.5))
```



Out[7]:
```
<sympy.plotting.plot.Plot at 0x94500b0>
```

3. (Ctd) Solving Equations Numerically (Approximate)

From the graph, y=50 when x is between 0.5 and 1.0 and also when x is about 3. So we can now use Python's **nsolve** command (numerically solve) by also including a starting "guess" near the solution

In [8]:
```
x1=nsolve(V-50,0.5)
x2=nsolve(V-50,3)
print('V=50 when x=',x1,x2)
```

```
V=50 when x= 0.591257469330590 3.10926620932673
```

These answers agree with the first two answers in VsolnAPPROX. What happened to the third solution? (HINT: look at the third solution and re-read the paragraph right before we graphed the Volume)

In summary, you should be able to use the following commands in this lab:

symbols

simplify or **expand** or **factor**

print

subs

evalf

for (list comprehension)

solve

plot

nsolve

In [ ]: