

Python Template for a Single Non-Linear Equation

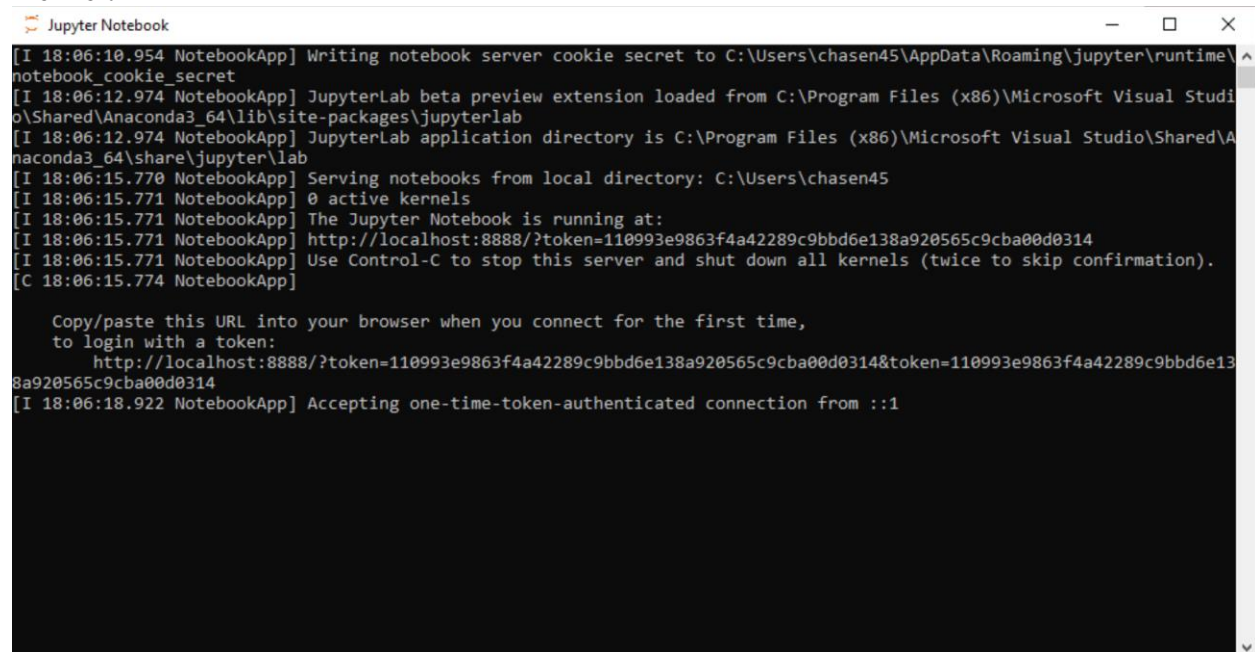
Prepared by Nick Chase

The use of this Python template for solving a steady state, adiabatic combustion reaction problem will be demonstrated using example 9.6-2 from the book of Felder and Rousseau: “Elementary Principles of Chemical Processes”. The problem will be modified to gain insightful knowledge on how to use Python to solve a single non-linear equation (NLE). This will require Citrix to open up the software called Jupyter Notebook. Below are the steps in opening Jupyter Notebook in Citrix and how to solve this problem.

Open and log into Citrix.

Once the Citrix display is loaded, type “Jupyter Notebook” into the search bar in the lower left corner of the screen. A pop-up window, shown below, will appear. **Do not close out the pop-up window, this serves as the Kernel to run the program.**

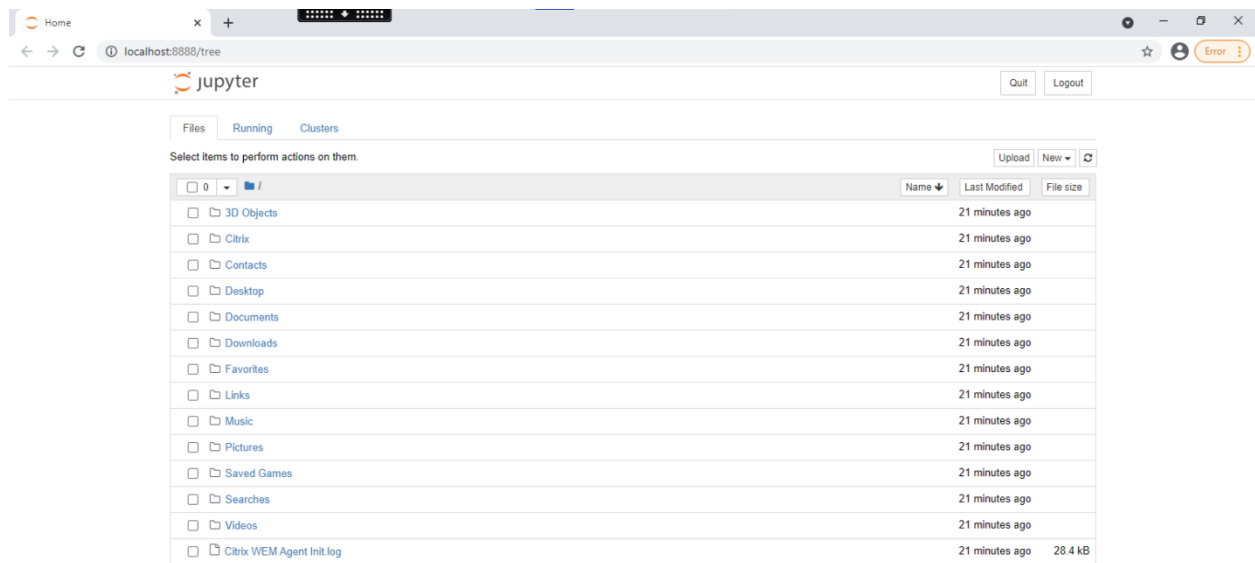
Kernel:

A screenshot of a terminal window titled "Jupyter Notebook". The window has a black background with white text. The text shows the Jupyter Notebook application starting up, including writing the cookie secret, loading the JupyterLab extension, and serving notebooks from a local directory. It also displays the URL to access the notebook and a message about accepting a one-time-token-authenticated connection.

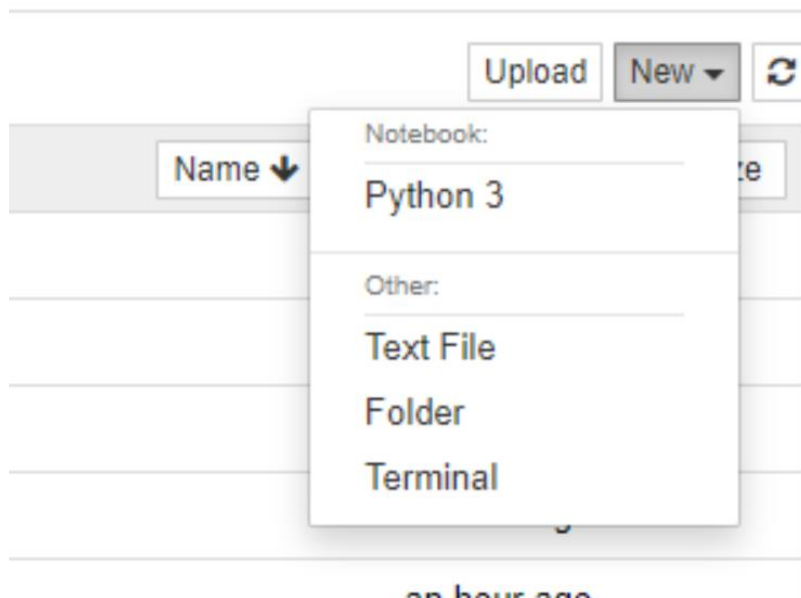
```
[I 18:06:10.954 NotebookApp] Writing notebook server cookie secret to C:\Users\chase45\AppData\Roaming\jupyter\runtime\notebook_cookie_secret
[I 18:06:12.974 NotebookApp] JupyterLab beta preview extension loaded from C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-packages\jupyterlab
[I 18:06:12.974 NotebookApp] JupyterLab application directory is C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\share\jupyter\lab
[I 18:06:15.770 NotebookApp] Serving notebooks from local directory: C:\Users\chase45
[I 18:06:15.771 NotebookApp] 0 active kernels
[I 18:06:15.771 NotebookApp] The Jupyter Notebook is running at:
[I 18:06:15.771 NotebookApp] http://localhost:8888/?token=110993e9863f4a42289c9bbd6e138a920565c9cba00d0314
[I 18:06:15.771 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 18:06:15.774 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8888/?token=110993e9863f4a42289c9bbd6e138a920565c9cba00d0314&token=110993e9863f4a42289c9bbd6e138a920565c9cba00d0314
[I 18:06:18.922 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

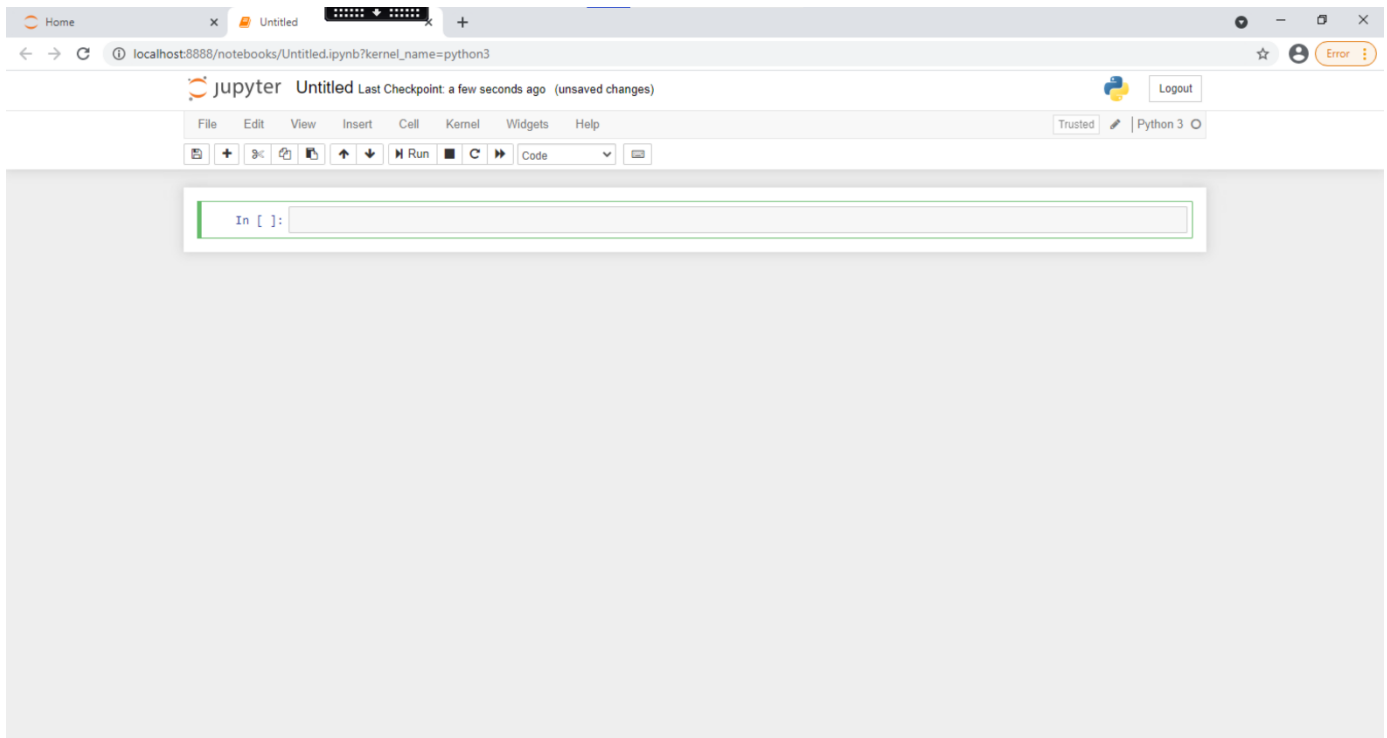
After that, the actual Jupyter Notebook program will load in Google Chrome.



Click on the pulldown menu called “new” and click on Python 3. This will start up a new notebook.



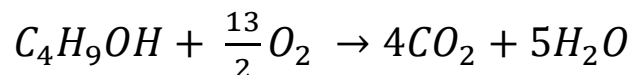
The image below is what your screen should look like now.



Now you can start the problem below.

Adiabatic Flame Combustion

The objective of this problem is to obtain an outlet temperature for a steady-state, adiabatic reactor that involves a full combustion reaction. This involves components of iso-butanol, air (oxygen and nitrogen), water and carbon dioxide. Below is the reaction happening inside the reactor.



Additionally, this problem requires 20% excess air to ensure that the reaction is a full combustion.

From the lecture videos, an adiabatic system means that the system does not give or lose energy, meaning $Q=0$. In deriving the mass and energy balance, the equation needed below is what should be obtained. Since a reaction is involved, the energy balance will need to be on a molar basis.

$$\sum F_{in_i} * \hat{H}_{in_i} = \sum F_{out_i} * \hat{H}_{out_i} + \overset{\text{Adiabatic}}{\cancel{Q}}$$

Where F is mol/s and \hat{H} is kJ/mol

The template underneath the images will explain the roles of each line or lines and will give you a further understanding on how to use programming and solving the problem.

In Jupyter Notebook which uses Python, you will need to import libraries to use functions or actions like a non-linear solver “fsolve” or some math functions like the square root (sqrt for short). The libraires being import are shown in the image below. Once your block of code looks like the one in the image, click CTRL + ENTER at the same time.

In [76]:

```
1 import math
2 import numpy as np
3 from scipy.optimize import fsolve
```

import math	Imports the math library to use basic math functions
import numpy as np	Imports the numpy library, acts similar to the math library
from scipy.optimize import fsolve	Imports fsolve from the scipy.optimize library to solve non-linear functions

If you want to know if the code ran then see if there is a number in between the brackets like in the image above. The number is 76

Coding is **spelling and case sensitive**. Be careful when typing in code because errors can occur as shown in the images below.

In [4]:

```
1 import math
2 import numpy as np
3 from scipy.Optimize import fsolve
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
<ipython-input-4-c3d27a620266> in <module>
      1 import math
      2 import numpy as np
----> 3 from scipy.Optimize import fsolve
```

```
ModuleNotFoundError: No module named 'scipy.Optimize'
```

“o” is uppercase, this needs to be lowercase.

```
In [6]: 1 import math
        2 import numpy as np
        3 from scipy.optimize import fsolve
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
<ipython-input-6-be8558afef98> in <module>
      1 import math
----> 2 import numpy as np
      3 from scipy.optimize import fsolve

ModuleNotFoundError: No module named 'numpy'
```

numpy is misspelled.

In this example the def was in upper case Def so the error is invalid syntax. Also notice that the def was not green! Python automatically makes this green in Jupyter Notebooks

```
In [7]: 1 Def NLEfun(T):
        2     O2Excess=1.2*(13/2) #mol/s
        3     NewN2=0.79/0.21*O2Excess #mol/s
        4     F1o=1#mol/s isobutanol
        5     F2o=O2Excess#mol/s O2
        6     F3o=NewN2#mol/s N2
40      Hdotin=H1Liq*F1o
41      Hdotout=H2Vap*F2+H3Vap*F3+H5Vap*F5+H4Vap*F4
42      fT= Hdotin-Hdotout
43      return fT
```

```
File "<ipython-input-7-cldff6b8b3d2>", line 1
Def NLEfun(T):
    ^
```

SyntaxError: invalid syntax

After running the first block of code, insert a new block which will define the equations needed to solve for the outlet temperature (adiabatic flame temperature).

Jupyter Untitled1 Last Checkpoint: 20 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
In [6]: 1 import math
        2 import numpy as np
        3 from scipy.optimize import fsolve
```

```
In [ ]: 1
```

Click here to insert a new code block.

The next page will show the entire code that is needed. There are some things to mention when looking at it. First, each line represents a new line of code. Usually in different programming languages, the line would end with a semi-colon. Matlab uses this to suppress printing the line whereas Python does not use it. Another important aspect is that wherever you see a '#' followed by characters, this is a comment. Note that comments are not run by the program and are there for the user to make notes to themselves or future readers. For example, when you see:

```
F4o=0#mol/s H2O
```

The program is only running:

```
F4o=0
```

And the comments such as, #mol/s H2O, are for people reading the code to get a better understanding of how the code works. In this case you are giving the units and specifying which chemical species is being modelled. It is always important to keep track of units since Python has no way to determine if the answer is in the correct units.

```

1 def NLEfun(T):
2     O2Excess=1.2*(13/2) #mol/s
3     NewN2=0.79/0.21*O2Excess #mol/s
4     F1o=1#mol/s isobutanol
5     F2o=O2Excess#mol/s O2
6     F3o=NewN2#mol/s N2
7     F4o=0#mol/s H2O
8     F5o=0#mol/s CO2
9     F1=F1o-1 #mol/s
10    F2=F2o-13/2#mol/s
11    F3=F3o-0 #mol/s
12    F4=F4o+5#mol/s
13    F5=F5o+4 #mol/s
14    H1formLq=-3.347e2#kJ/mol
15    H1formVap=-2.832e2#kJ/mol
16    H2formVap=0
17    H3formVap=0
18    H4formVap=-2.418e2#kJ/mol
19    H5formVap=-3.9351e2#kJ/mol
20    To=25+273.15 Inlet Temperature in Kelvin
21    Cp1=(87940,241600,1718,165400,798.7)
22    Cp2=(29103,10040,2526.5,9356,1153.8)
23    Cp3=(29105,8614.9,1701.6,103.47,909.79)
24    Cp4=(33363,26790,2610.5,8896,1169)
25    Cp5=(29370,34540,1428,26400,588)
26    Cp2=1/1e6*(Cp2[0]*(T-To)+Cp2[1]*Cp2[2]*(1/np.tanh(Cp2[2]/T)-1/np.tanh(Cp2[2]/To))
27    -Cp2[3]*Cp2[4]*(np.tanh(Cp2[4]/T)-np.tanh(Cp2[4]/To)))
28    Cp3=1/1e6*(Cp3[0]*(T-To)+Cp3[1]*Cp3[2]*(1/np.tanh(Cp3[2]/T)-1/np.tanh(Cp3[2]/To))
29    -Cp3[3]*Cp3[4]*(np.tanh(Cp3[4]/T)-np.tanh(Cp3[4]/To)))#kJ/mol
30    Cp4=1/1e6*(Cp4[0]*(T-To)+Cp4[1]*Cp4[2]*(1/np.tanh(Cp4[2]/T)-1/np.tanh(Cp4[2]/To))
31    -Cp4[3]*Cp4[4]*(np.tanh(Cp4[4]/T)-np.tanh(Cp4[4]/To)))#kJ/mol
32    Cp5=1/1e6*(Cp5[0]*(T-To)+Cp5[1]*Cp5[2]*(1/np.tanh(Cp5[2]/T)-1/np.tanh(Cp5[2]/To))
33    -Cp5[3]*Cp5[4]*(np.tanh(Cp5[4]/T)-np.tanh(Cp5[4]/To)))#kJ/mol
34    H1Liq=H1formLq#kJ/mol
35    H2Vap=H2formVap+Cp2#kJ/mol
36    H3Vap=H3formVap+Cp3#kJ/mol
37    H4Vap=H4formVap+Cp4#kJ/mol
38    H5Vap=H5formVap+Cp5#kJ/mol
39    Hdotin=H1Liq*F1o
40    Hdotout=H2Vap*F2+H3Vap*F3+H5Vap*F5+H4Vap*F4
41    fT= Hdotin-Hdotout
42    return fT

```

Inlet molar flowrates

Each number refers to a chemical species given in the comments

Outlet molar flowrates with changes from reaction table

Standard heats of formation for liquid and vapor

“formLq” is standard heat of formation for liquid

“formVap” is standard heat of formation for vapor

Cpv coefficients obtained from DIPPR. Each is listed in an array as (A,B,C,D,E)

This will be explained in the table below

Specific molar enthalpy values

H1Liq only needs heat of formation it is a liquid at 25°C

Enthalpy inlet and outlet flows from energy balance

These lines of code will be explained in the table next page

NOTE: In order for this to run, anything after the “def NLEfun(T):” must be indented, the code will automatically indent after that function. You should copy the left column into a Jupyter Notebook cell.

<pre>def NLEfun(T):</pre>	<p>This defines the non-linear block function, which will hold the equations and constants that will be needed to solve the non-linear function. The T in parenthesis stands as the outlet temperature being the only unknown for this problem. This function, NLEfun, will be called later in the next code block.</p> <p>In this function, ‘T’ is what we call the input parameter. When using the function in the future, T will be a user defined value that you can input into the function. This will be further explained later.</p>
<pre>def NLEfun(T): O2Excess=1.2*(13/2) #mol/s NewN2=0.79/0.21*O2Excess #mol/s F1o=1#mol/s isobutanol F2o=O2Excess#mol/s O2 F3o=NewN2#mol/s N2 F4o=0#mol/s H2O F5o=0#mol/s CO2 F1=F1o-1 #mol/s F2=F2o-13/2#mol/s F3=F3o-0 #mol/s F4=F4o+5#mol/s F5=F5o+4 #mol/s H1formLq=-3.347e2#kJ/mol H1formVap=-2.832e2#kJ/mol H2formVap=0 H3formVap=0 H4formVap=-2.4182e2#kJ/mol H5formVap=-3.9351e2#kJ/mol Tref=25+273.15</pre>	<p>These are explicit equations that were explained above.</p> <p>Constants are defined.</p> <p>For students unfamiliar with coding, the use of an equal sign ‘=’ is an assignment operator. F1o=1 Assigns the value 1.0 to the variable Flo.</p>
<pre>Cpv1=(87940,241600,1718,165400,798.7) Cpv2=(29103,10040,2526.5,9356,1153.8) Cpv3=(29105,8614.9,1701.6,103.47,909.79) Cpv4=(33363,26790,2610.5,8896,1169) Cpv5=(29370,34540,1428,26400,588)</pre>	<p>The heat capacity constants in the equation below are put in arrays. These constants are in order (A,B,C,D,E) as used in the equation below. In Python, an array of a list of values uses an index system. This index system always starts at a value of 0. For example, if you type print(Cpv1[0]), the value Python prints out would be 87940. print(Cpv3[3]) = 103.47</p>

The DIPPR Ideal Gas Heat Capacity equation is given as:

$$C_{pi} = A + B \left[\frac{C/T}{\sinh(C/T)} \right]^2 + D \left[\frac{E/T}{\cosh(E/T)} \right]^2 \text{ with T in Kelvin and Cp [=] J/kmol * K}$$

The integrated form is given as

$$\Delta \hat{H}_i = \int_{T_1}^{T_2} C_{p_i} dT = A\Delta T + BC \left[\coth\left(\frac{C}{T_2}\right) - \coth\left(\frac{C}{T_1}\right) \right] - DE \left[\tanh\left(\frac{E}{T_2}\right) - \tanh\left(\frac{E}{T_1}\right) \right]$$

And in python we use

$$\Delta \hat{H}_i = \int_{T_1}^{T_2} C_{p_i} dT = A\Delta T + BC \left[\frac{1}{\tanh\left(\frac{C}{T_2}\right)} - \frac{1}{\tanh\left(\frac{C}{T_1}\right)} \right] - DE \left[\tanh\left(\frac{E}{T_2}\right) - \tanh\left(\frac{E}{T_1}\right) \right]$$

Cp2=1/1e6*(Cpv2[0]*(T-Tref)+Cpv2[1]*Cpv2[2]*(1/np.tanh(Cpv2[2]/T)-1/np.tanh(Cpv2[2]/Tref)) -Cpv2[3]*Cpv2[4]*(np.tanh(Cpv2[4]/T)-np.tanh(Cpv2[4]/Tref))) Cp3=1/1e6*(Cpv3[0]*(T-Tref)+Cpv3[1]*Cpv3[2]*(1/np.tanh(Cpv3[2]/T)-1/np.tanh(Cpv3[2]/Tref)) -Cpv3[3]*Cpv3[4]*(np.tanh(Cpv3[4]/T)-np.tanh(Cpv3[4]/Tref)))#kJ/mol Cp4=1/1e6*(Cpv4[0]*(T-Tref)+Cpv4[1]*Cpv4[2]*(1/np.tanh(Cpv4[2]/T)-1/np.tanh(Cpv4[2]/Tref)) -Cpv4[3]*Cpv4[4]*(np.tanh(Cpv4[4]/T)-np.tanh(Cpv4[4]/Tref)))#kJ/mol Cp5=1/1e6*(Cpv5[0]*(T-Tref)+Cpv5[1]*Cpv5[2]*(1/np.tanh(Cpv5[2]/T)-1/np.tanh(Cpv5[2]/Tref)) -Cpv5[3]*Cpv5[4]*(np.tanh(Cpv5[4]/T)-np.tanh(Cpv5[4]/Tref)))#kJ/mol		
	The lines above are the integrated DIPPR Cp equations. As shown, each array value is called from its corresponding index value and array. Currently, there is no library that has the geometry function “coth”, this is substituted by 1/tanh(x) which is equal to coth(x). In the code, tanh is written as np.tanh, this is calling the tanh from the library that was imported in the first code block.	
H1Liq=H1formLq#kJ/mol H2Vap=H2formVap+Cp2#kJ/mol H3Vap=H3formVap+Cp3#kJ/mol H4Vap=H4formVap+Cp4#kJ/mol H5Vap=H5formVap+Cp5#kJ/mol	Additional explicit equation that were shown above, these are the total molar enthalpy values.	
Hdotin=H1Liq*F1o Hdotout=H2Vap*F2+H3Vap*F3+H5Vap*F5+H4Vap*F4	Inlet and outlet enthalpy flows, these were defined from the energy balance.	
fT= Hdotin-Hdotout	This is the energy balance, we know that for this problem that Inlet-Outlet=0 for steady state. In order to get fT=0, the outlet T value will need to be calculated.	
return fT	This will return the value fT calculated. This is the indication of the end of the defining function. So now that the function is defined, you may use NLEfun(T), and that single line will run this whole function and return the value of fT based on what value for T you input.	

After this code block is typed, press CTRL + ENTER to run the code. No results will be shown yet, this will happen in the last code block which is on the next page.

In Microsoft Excel, you have learned to use solver to make the cell like equation fT mentioned above, 0. Python will do the same thing with a few lines of code and a push of a button. The image below will display the result of the outlet temperature in Kelvin and Celsius.

```
1 Tguess=500
2 sol=fsolve(NLEfun,Tguess)
3 print(sol)
4 Tad=sol-273.15
5 print(Tad)
```

Tguess=500	This is the initial guess that you will provide. Knowing that this is a combustion reaction and an adiabatic system, you should assume that the temperature is higher than the initial temperature of 298.15K. The initial guess assumed will be 500 for this problem.
sol=fsolve(NLEfun,Tguess)	This line will calculate the outlet temperature labeled “sol”. The fsolve function will take the labeled defined code block above, the initial guess for outlet temperature and automatically calculate until $fT = 0$. The answer will not be shown until after the print statements are typed, shown on the next line.
print(sol)	This will print out the outlet temperature calculated from the fsolve function, the units for this answer would be in Kelvin.
Tad=sol-273.15	This is taking sol and subtracting 273.15, which in this case would be degrees Celsius now.
print(Tad)	As before, this will print out Tad which would be the final answer to the problem in Celsius.

```
[1996.48638165]
```

```
[1723.33638165]
```

This is what you should get from the print functions. You have now completed this Python tutorial! If you have any comments or issues/problems, please don't hesitate to email chasen45@students.rowan.edu