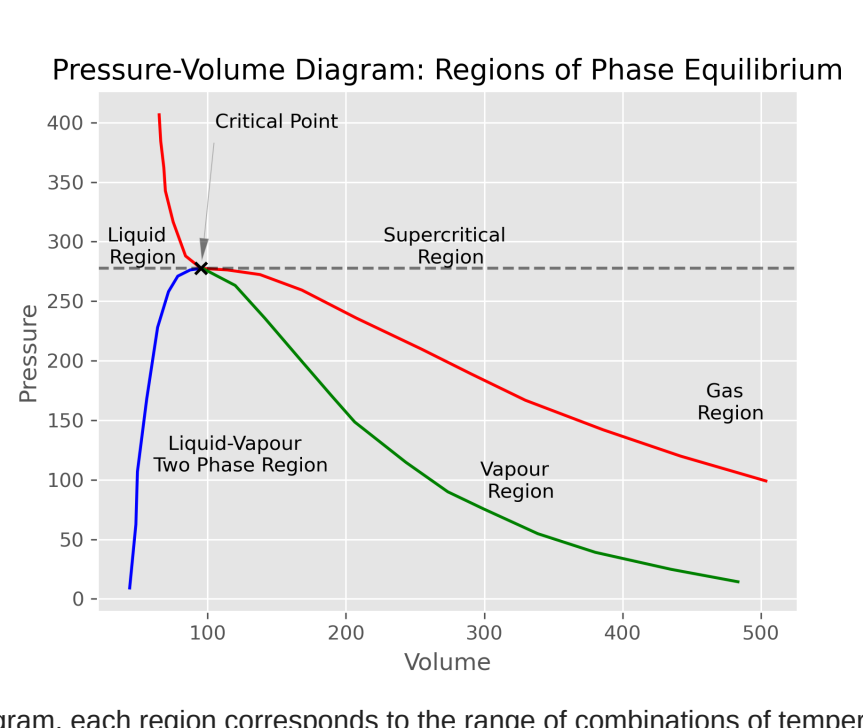
Thermo B - Student Exercises

Pressure-Volume Diagram



In a pressure-volume diagram, each region corresponds to the range of combinations of temperature and pressure over which that phase is stable 1

Liquid Region: The liquid region would be where the substance is in its liquid phase at a given combination of temperature and pressure.

Liquid-Vapor Region: The liquid-vapor region on a pressure-volume diagram represents the range of values where both liquid and vapor phases can coexist at equilibrium

Vapour Region: The vapour region would be where the substance is in its vapour phase at a given combination of temperature and pressure.

Gas Region: The gas region would be where the substance is in its gas phase at a given combination of temperature and pressure.

Supercritical Region: In a pressure-temperature phase diagram, the supercritical region is where the liquid and gas phases disappear to become a single supercritical phase 1. This occurs at conditions beyond the critical point, where temperature and pressure are both higher than their respective critical values.

Ideal Gas Law

The ideal gas law is a mathematical expression of the state of a hypothetical ideal gas, derived from the kinetic theory of gases and named by the physicist Jacques Charles in 1787 1. It is a special case of the general gas law, which describes the state of any gas.

$$PV = nRT$$

where:

- P is the pressure of the gas
- V is the volume of the gas
- n is the number of moles of gas
- R is the ideal gas constant
- T is the temperature of the gas

Task 1: Idea Gas Law Calculation

Calculate the theoretical pressure of the gas, CO_2, at various temperatures and volumes.

> **Python**: For this task, you will need to use the ideal gas law function from the `thermo` module. The function is called `ideal_gas_law` and takes the following arguments: `ideal_gas_law(gas_constant: float, list_of_volumes: List, list_of_temperatures: List)`. The function returns a Dataframe with the calculated pressures for each combination of temperature and volume. The index of the Dataframe is the volume and the columns are the temperatures.

In [2]:
```
# Press Shift+Enter to run the code
from src.thermo import ideal_gas_law
```

> ***Tip***: You can get the docstring for a function by using the ? symbol after the function name

In [3]:
```
# Constant values for the ideal gas law function
## R is the gas constant
## list_of_volumes is a list of volumes in m^3 - You will need to create this list, ranging from 30
## list_of_temperatures is a list of temperatures in K - You will need to create this list. The val

R = 8.314 # cm^3 MPa / K mol
list_of_volumes = [] # cm^3
list_of_temperatures = [] # K
```

In [4]:
```
### Teacher Solution

R = 8.314 # J/mol*K
list_of_volumes = list(range(30, 601)) # m^3
list_of_temperatures = [255.2, 265.1, 274.4, 284.0, 294.4, 304.1, 334.1, 354.1] # K
```

In [5]:
```
# Call the ideal_gas_law function with the appropriate arguments and set it to the variable ideal_g
%%script false --no-raise-error
ideal_gas_law_df = ideal_gas_law()
```

```
UsageError: Line magic function `%%script` not found.
```

In [6]:
```
### Teacher Solution

ideal_gas_law_df = ideal_gas_law(R, list_of_volumes, list_of_temperatures)
ideal_gas_law_df
```

Out[6]:

	255.2	265.1	274.4	284.0	294.4	304.1	334.1	354.1
30	70.724427	73.468047	76.045387	78.705867	81.588053	84.276247	92.590247	98.132913
31	68.442994	71.098110	73.592310	76.166968	78.956181	81.557658	89.603465	94.967335
32	66.304150	68.876294	71.292550	73.786750	76.488800	79.008981	86.803356	91.999606
33	64.294933	66.789133	69.132170	71.550788	74.170958	76.614770	84.172952	89.211739
34	62.403906	64.824747	67.098871	69.446353	71.989459	74.361394	81.697276	86.587865
...	...	...	...	...	...	...	...	...
596	3.559954	3.698056	3.827788	3.961705	4.106781	4.242093	4.660583	4.939576
597	3.553991	3.691862	3.821376	3.955069	4.099902	4.234987	4.652774	4.931302
598	3.548048	3.685688	3.814986	3.948455	4.093046	4.227905	4.644996	4.923056
599	3.542125	3.679535	3.808617	3.941863	4.086213	4.220847	4.637241	4.914837
600	3.536221	3.673442	3.802269	3.935293	4.079403	4.213812	4.629512	4.906646

571 rows × 8 columns

Task 2: Ideal Gas Law Plot and Critical Point

In this task you will need to on the same plot:

1. Plot the pressure of the gas, CO_2, at various temperatures and volumes using the Dataframe from Task 1.
2. Plot the critical point of the gas, CO_2, on the pressure-volume diagram.

> **Python**: For this task, you will need to use the `plot_ideal_gas_law` function from the `thermo` module. The function is called `plot_ideal_gas_law` and takes the following arguments: `plot_ideal_gas_law(ideal_gas_law_df: pd.DataFrame)`. The function returns a plot of the ideal gas law Dataframe. However, the plot will not be shown in the notebook. You will need to use the `plt.show()` function to display the plot. Additionally, you will need to use the `plt.figure(figsize=(width, height))` function to set the size of the plot. Recommended is a width of 10 and a height of 10. You will also need to use the `plt.title()` function to set the title of the plot. You can use the `plt.xlabel()` and `plt.ylabel()` functions to set the x and y axis labels.

> ***Tip***: Before calling `plt.show()`, ensure you have plotted both the ideal gas law and the critical point on the same plot.

> ***Tip***: You can get the docstring for a function by using the ? symbol after the function name

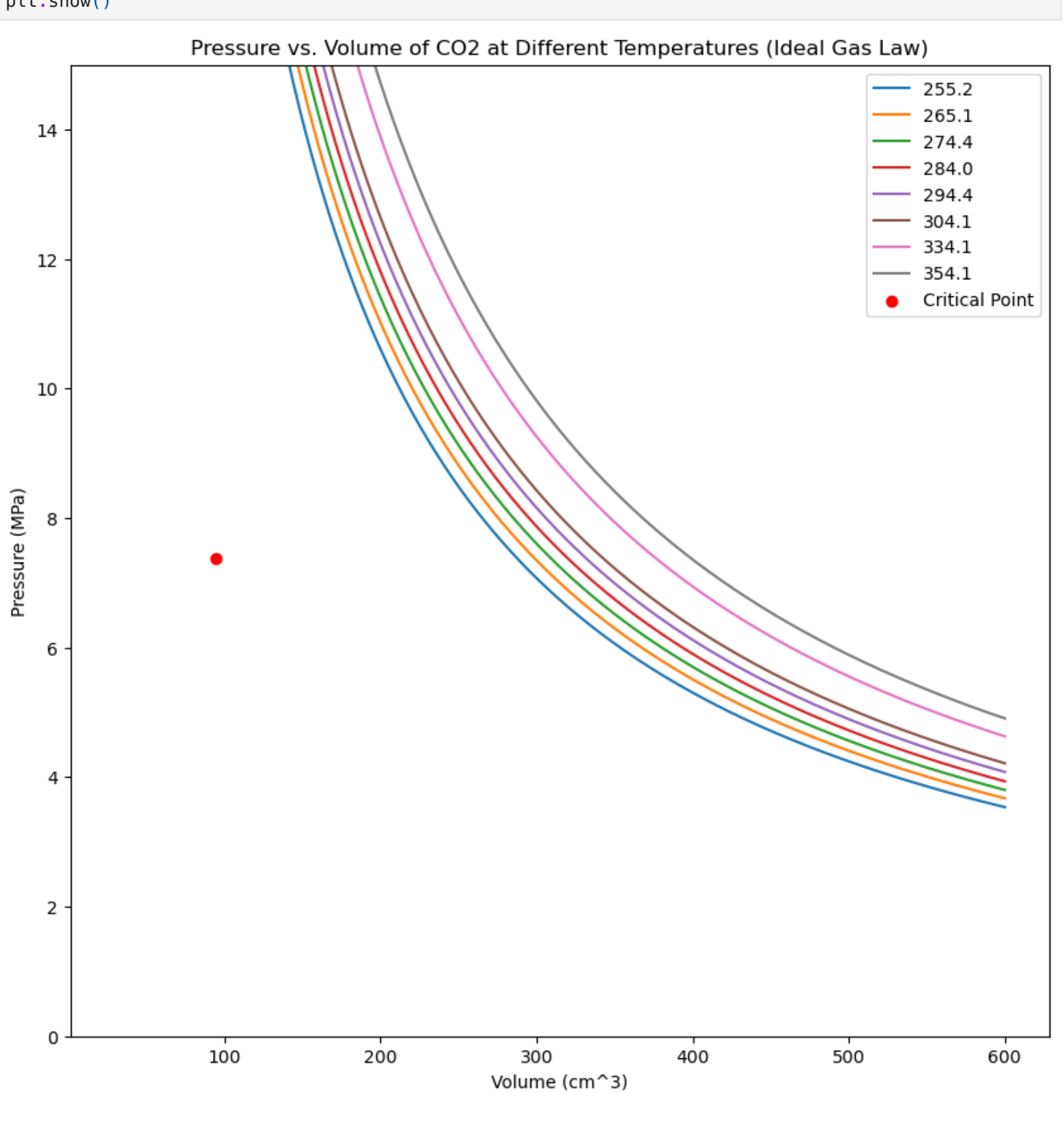
In [7]:
```
# Press Shift+Enter to run the code
from src.thermo import plot_ideal_gas_law
import matplotlib.pyplot as plt
```

In [8]:
```
# From literature of CO2 Critical Point
critical_point_pressure = 7.377 # MPa
critical_point_temperature = 304.1 # K
critical_point_volume = 94.120 # cm^3
```

In [9]:
```
# Call the plot_ideal_gas_law function with the appropriate argument - You will not need to set it
#
%%script false --no-raise-error
plot_ideal_gas_law()
```

```
UsageError: Line magic function `%%script` not found.
```

In [10]:
```
### Teacher Solution

plt.figure(figsize=(10, 10))

plot_ideal_gas_law(ideal_gas_law_df)

plt.xlabel('Volume (cm^3)')
plt.ylabel('Pressure (MPa)')

plt.scatter(critical_point_volume, critical_point_pressure, color='red', label='Critical Point')

plt.legend()
plt.title('Pressure vs. Volume of CO2 at Different Temperatures (Ideal Gas Law)')

plt.show()
```



As it can be seen, the Critical Point of CO_2 is nowhere near the ideal gas law for temperature 304.1K

Assumptions:

- The gas is a perfect gas
- The volume of the gas is negligible compared to the volume of the container
- There is no intermolecular attraction or repulsion between the gas molecules

Limitations:

- The ideal gas law is only valid for a small range of temperatures and pressures
- It is unrealiable at high pressures and low temperatures
- Cannot predict vapur-liquid coexistence

Peng-Robinson Equation of State

Peng-Robinson equation of state

$$P = \frac{RT}{V-b} - \frac{a}{V(V+b)+b(V-b)} \qquad a = 0.45724\frac{\alpha R^2 T_c^2}{P_c} \qquad b = 0.07780\frac{RT_c}{P_c}$$

$$\alpha = \left(1+\kappa(1-\sqrt{T/T_c})\right)^2 \qquad \kappa = 0.37464 + 1.54226\omega - 0.26992\omega^2$$

where P is the pressure (Pa);
V is the molar volume (m³ mol⁻¹);
R is the gas constant (8.314 J mol⁻¹ K⁻¹);
T is the absolute temperature (K);
P_c is the critical pressure for the component of interest (Pa);
T_c is the critical temperature for the component of interest (K);
and ω is the acentric factor for the component of interest.

The Peng-Robinson equation of state (PR EOS) is a cubic equation of state that was developed in 1976 at The University of Alberta by Ding-Yu Peng and Donald Robinson 1. It is used to predict the behavior of fluids, particularly natural gas systems 2.

The PR EOS is more accurate than the Ideal Gas Law for predicting the behavior of real gases because it takes into account molecular interactions and volume 3.

Task 3: Peng-Robinson Equation of State Calculation

Calculate the theoretical pressure of the gas, CO_2, at various temperatures and volumes using the Peng-Robinson Equation of State.

> **Python**: For this task, you will need to use the Peng-Robinson Equation of State function from the `thermo` module. The function is called `peng_robinson_eos` and takes the following arguments: `peng_robinson_eos(gas_constant: float, list_of_volumes: List, list_of_temperatures: List, kappa: float, bc: float, ac: float, alpha: Callable)`. The function returns a Dataframe with the calculated pressures for each combination of temperature and volume. The index of the Dataframe is the volume and the columns are the temperatures.

In [11]:
```
from src.thermo import peng_robinson_eos
```

In [12]:
```
# Values for the Peng-Robinson EOS
critical_point_pressure = 7.377 # MPa
critical_point_temperature = 304.1 # K
w = 0.224 # Acentric Factor of CO2 - Acquired from literature
```

Kappa

The kappa parameter is a dimensionless parameter that is calculated from the acentric factor of the substance. The acentric factor is used to measure the non-sphericity of a molecule. The kappa parameter is calculated using the following equation:

$$\kappa = 0.37464 + 1.54226\omega - 0.26992\omega^2$$

where:

- ω is the acentric factor of the substance

In [13]:
```
# Find Kappa for the Peng-Robinson EOS and set the value to the variable kappa
%%script false --no-raise-error
kappa = None # < - Enter your code here, remove None
```

```
UsageError: Line magic function `%%script` not found.
```

In [14]:
```
### Teacher Solution
kappa = 0.37464 + 1.54226 * w - 0.26992 * w**2
```

Alpha

The alpha function is a function of temperature that is used to calculate the acentric factor of the gas. The acentric factor is a measure of the deviation of the gas from ideal gas behavior. The acentric factor is used to calculate the a and b constants of the Peng-Robinson Equation of State. The alpha function is defined as:

$$\alpha(T) = \left(1 + \kappa\left(1 - \sqrt{\frac{T}{T_c}}\right)\right)^2$$

where:

- T is the temperature
- κ is the acentric factor
- T_c is the critical temperature

As we already know κ and T_c, we can create a function that takes in T and returns the value of alpha:

Two ways to do this - *lambda* or defining a *function*

You can run either Cell below

In [15]:
```
# Lambda version of alpha
alpha = lambda T: (1 + kappa * (1 - (T / critical_point_temperature)**0.5))**2
```

In [16]:
```
# Function version of alpha
def alpha(T):
    return (1 + kappa * (1 - (T / critical_point_temperature)**0.5))**2
```

In [17]:
```
%%script false --no-raise-error
peng_robinson_eos_df = peng_robinson_eos()
```

a variable

The a variable is defined as:

$$a = 0.45724\frac{\alpha R^2 T_c^2}{P_c}$$

where:

- α is the alpha function
- R is the ideal gas constant
- T_c is the critical temperature
- P_c is the critical pressure

In [18]:
```
# Since alpha is a function/lambda, it will not be included in the variable a, but instead will be
%%script false --no-raise-error
a = None
```

```
UsageError: Line magic function `%%script` not found.
```

In [19]:
```
### Teacher Solution
a = 0.45724 * (R * critical_point_temperature)**2 / critical_point_pressure
```

b variable

The b variable is defined as:

$$b = 0.07780\frac{RT_c}{P_c}$$

where:

- R is the ideal gas constant
- T_c is the critical temperature
- P_c is the critical pressure

In [20]:
```
# Find the value of b and set it to the variable b
b = None # < - Enter your code here, remove None
```

In [21]:
```
### Teacher Solution
b = 0.07780 * R * critical_point_temperature / critical_point_pressure
```

Peng-Robinson Equation of State Function

Now that you have the required variables for the Peng-Robinson Equation of State function, you can create the function.

In [22]:
```
# Call the peng_robinson_eos function with the appropriate arguments and set it to the variable per
%%script false --no-raise-error
peng_robinson_eos_df = peng_robinson_eos()
peng_robinson_eos_df
```

```
UsageError: Line magic function `%%script` not found.
```

In [23]:
```
### Teacher Solution
peng_robinson_eos_df = peng_robinson_eos(R, list_of_volumes=list_of_volumes, list_of_temperatures=l
peng_robinson_eos_df
```

Out[23]:

	255.2	265.1	274.4	284.0	294.4	304.1	334.1	354.1
30	387.495310	417.969341	446.437831	475.671860	507.175837	536.411348	625.997372	685.083781
31	255.738637	280.173799	302.978943	326.376115	351.566816	374.922874	446.375067	493.410557
32	177.488559	198.052113	217.228878	236.888901	258.040023	277.636410	337.505860	376.854042
33	126.915275	144.759859	161.390218	178.429183	196.748975	213.711859	265.477297	299.454065
34	92.348148	108.163258	122.894214	137.979319	154.189840	169.192022	214.930578	244.917854
...	...	...	...	...	...	...	...	...
596	2.575772	2.747204	2.907512	3.072284	3.250016	3.415101	3.921813	4.256669
597	2.572939	2.744031	2.904022	3.068469	3.245852	3.410613	3.916337	4.250543
598	2.570111	2.740865	2.900540	3.064663	3.241698	3.406137	3.910876	4.244434
599	2.567288	2.737705	2.897066	3.060866	3.237555	3.401672	3.905429	4.238343
600	2.564471	2.734551	2.893599	3.057078	3.233421	3.397218	3.899998	4.232270

571 rows × 8 columns

Task 4: Peng-Robinson Equation of State Plot

In this task you will need to on the same plot:

1. Plot the pressure of the gas, CO_2, at various temperatures and volumes using the Dataframe from Task 3
2. Plot the critical point of the gas, CO_2 on the pressure-volume diagram

> **Python**: For this task, you will need to use the `plot_peng_robinson_eos` function from the `thermo` module. The function takes the following arguments: `plot_peng_robinson_eos(peng_robinson_eos_df: pd.DataFrame)`. The function returns a plot of the ... However, the plot will not be shown in the notebook. You will need to use the `plt.show()` function to display the plot. Additionally, you will need to use the `plt.figure(figsize=(width, height))` function to set the size of the plot. Recommended is a width of 10 and a height of 10. You will also need to use the `plt.title()` function to set the title of the plot. You can use the `plt.xlabel()` and `plt.ylabel()` functions to set the x and y axis labels.

> ***Tip***: Before calling `plt.show()`, ensure you have plotted both the ideal gas law and the critical point on the same plot.

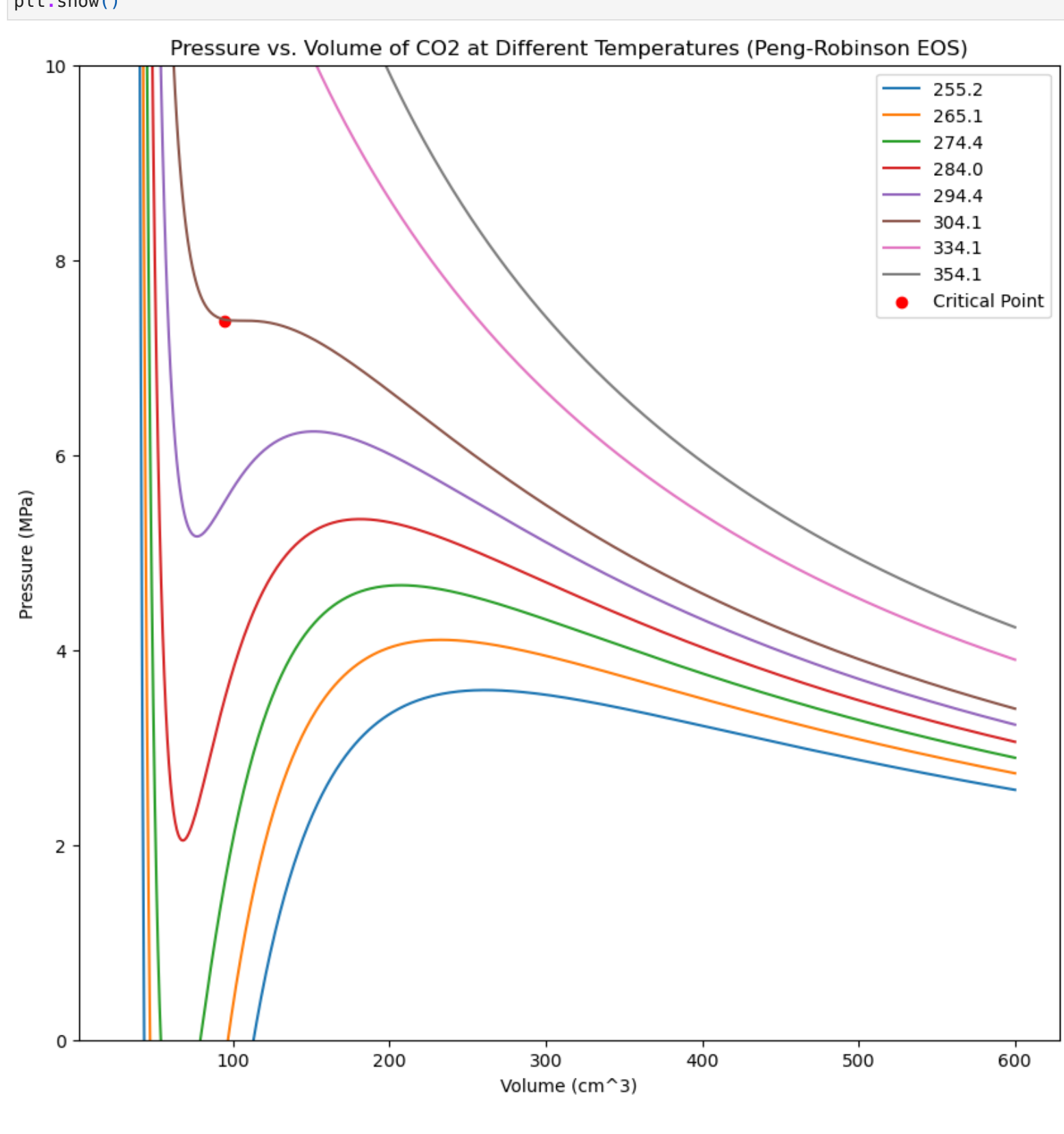
> ***Tip***: You can get the docstring for a function by using the ? symbol after the function name

In [24]:
```
from src.thermo import plot_peng_robinson_eos
```

In [25]:
```
# Call the plot_ideal_gas_law function with the appropriate argument - You will not need to set it
#
%%script false --no-raise-error
plot_peng_robinson_eos()
```

```
UsageError: Line magic function `%%script` not found.
```

In [26]:
```
### Teacher Solution

plt.figure(figsize=(10, 10))
plt.ylim(0, 10)
plot_peng_robinson_eos(peng_robinson_eos_df)

plt.xlabel('Volume (cm^3)')
plt.ylabel('Pressure (MPa)')

plt.scatter(critical_point_volume, critical_point_pressure, color='red', label='Critical Point')

plt.legend()
plt.title('Pressure vs. Volume of CO2 at Different Temperatures (Peng-Robinson EOS)')

plt.show()
```



Peng-Robinson Equation of State Plot

As you can see, the Peng-Robinson Equation of State is a much better fit for the data than the Ideal Gas Law.

It procies better accuracy near the critical point and for liquid molar volumes and it can be used to predict the vapour-liquid equilibria with good accuracy when combined with appropriate mixng rules.

Peng-Robinson Equation of State Polynomial Form

Peng-Robinson in polynomial form

$$Z^3 - (1-B)Z^2 + (A - 3B^2 - 2B)Z - (AB - B^2 - B^3) = 0$$

$$A = \frac{aP}{R^2T^2} \qquad B = \frac{bP}{RT} \qquad Z = \frac{PV}{RT}$$

where Z is the compressibility factor;
P is the pressure (Pa);
V is the molar volume (m³ mol⁻¹);
R is the gas constant (8.314 J mol⁻¹ K⁻¹);
T is the absolute temperature (K);
a is the Peng-Robinson attraction parameter;
and b is the Peng-Robinson covolume.

In a more compact and dimensionless form, the Peng-Robinson EOS can be restated as a cubic in Z. This is the polynomial form of the Peng Robinson Equation of State. This sets up a cubic equation in Z, which has three real roots (as opposed to being imaginary). The largest root is the value of Z_v; the smallest root is the value of Z_L; and the third root is discarded as having no physical meaning.

So how do we solve for the roots? In Python, we will use the module `CubicEquationSolver` which contains the function `solve(a, b, c, d)` which takes the coefficients of the cubic equation and returns the roots in an array.

Polynomial Structure:

$$ax^3 + bx^2 + cx + d = 0$$

Task 5: Finding the Roots of the Peng Robinson Equation of State

Find the roots of the Peng Robinson Equation of State Polynomial for the gas, CO_2, at the temperature 284.0K and pressure at 4.6 MPa.

In [27]:
```
# Shift + Enter to run the code
import CubicEquationSolver as ces
```

Breaking up the equation: Z^3

Define the variable Z_3 as the coefficient of Z^3 from the polynomial

In [28]:
```
Z_3 = None # < - Enter your code here, remove None
```

In [29]:
```
### Teacher Solution
Z_3 = 1
```

Breaking up the equation: Z^2

Define the variable Z_2 as the coefficient of Z^2 from the polynomial

> ***Tip***: $-(1-B)$

> ***Tip***: $B = \frac{bP}{RT}$

> ***Tip***: Use the variable b from the previous task

In [30]:
```
Z_2 = None # < - Enter your code here, remove None
```

In [31]:
```
### Teacher Solution
pressure = 4.6 # MPa
temperature = 284 # K

B = b * pressure / (R * temperature)
Z_2 = -1+ B
print (Z_2)
```

```
-0.9480535677123606
```

Breaking up the equation: Z^1

Define the variable Z_1 as the coefficient of Z^1 from the polynomial

> ***Tip***: $(A - 2B - 3B^2)$

> ***Tip***: $A = \frac{aP}{R^2T^2}$

> ***Tip**: a is the variable from the previous task BUT it must be adjusted due to the alpha function, which is a function of T!*

In [32]:
```
alpha_temp = None # < - Enter your code here, remove None
a = None # < - Enter your code here, remove None
A = None # < - Enter your code here, remove None

Z_1 = None # < - Enter your code here, remove None
```

In [33]:
```
### Teacher Solution

alpha_temp = alpha(temperature)
a = 0.45724 * (R * critical_point_temperature)**2 / critical_point_pressure * alpha_temp

A = a * pressure / (R * temperature)**2

Z_1 = A -2 * B - 3 * B**2
print (Z_1)
```

```
0.23062666623811
```

Breaking up the equation: Z^0

Define the variable Z_0 as the coefficient of Z_O from the polynomial

> ***Tip***: $-(AB - B^2 - B^3)$

In [34]:
```
Z_0 = None # < - Enter your code here, remove None
```

In [35]:
```
### Teacher Solution
Z_0 = B**3 + B**2 - A * B
print (Z_0)
```

```
-0.014959012141287458
```

Solving for the roots

Recall that you will need to use the `CubicEquationSolver` module to solve for the roots. The function is `solve(a, b, c, d)` where it takes the coefficients of the polynomial. Furthermore, the return value is an array of the roots. As there are three roots, you will need to store each root in a variable. The variables should be called:

Z_v: The largest root

Z_L: The smallest root

Z_none: The middle root

In [36]:
```
%%script false --no-raise-error
_, _, _ = ces.solve() # < - Enter your code here. Variables will need to be set

print (f'Vapour root: {_},\nNo Physical Meaning root: {_},\niquid root: {_}')
```

In [37]:
```
### Teacher Solution

Z_v, Z_l, Z_none = ces.solve(Z_3, Z_2, Z_1, Z_0)
print (f'Vapour root: {Z_v},\nNo Physical Meaning root: {Z_none},\nLiquid root: {Z_l}')
```

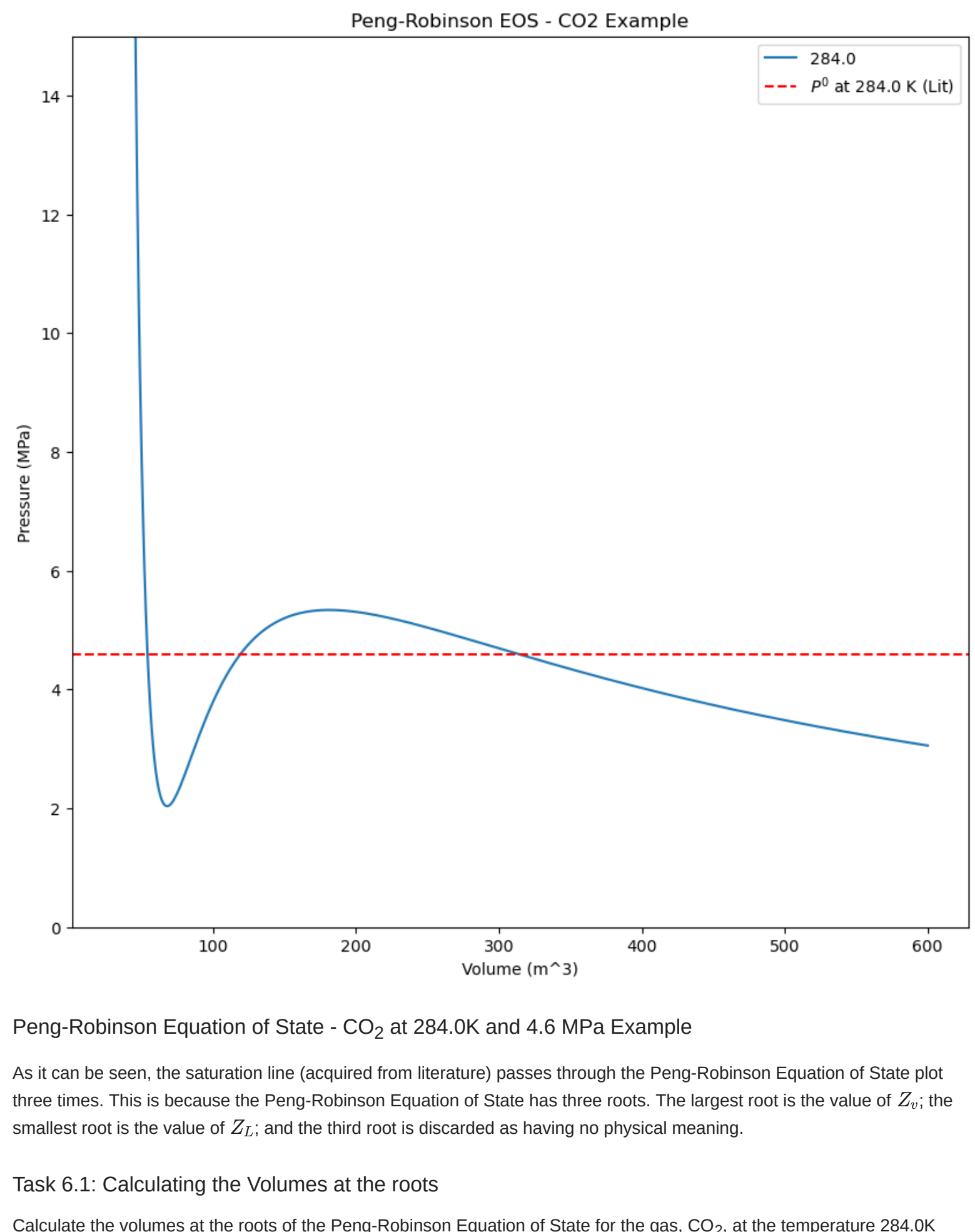
```
Vapour root: 0.6103487677467386,
No Physical Meaning root: 0.23211514781296316,
Liquid root: 0.10558965215265873
```

Task 6: Peng-Robinson Equation of State Plot - Roots

Now let's take a look at the plot of CO_2 at temperature 284.0K and pressure at 4.6 MPa. On the plot we will also plot the saturation pressure and the saturation temperature, acquired from literature.

In [38]:
```
import matplotlib.pyplot as plt
from src.thermo import plot_example_ideal_gas_law_and_peng_robinson_eos
```

In [39]:
```
plt.figure(figsize = (10, 10))

plot_example_ideal_gas_law_and_peng_robinson_eos()
```



Peng-Robinson Equation of State - CO_2 at 284.0K and 4.6 MPa Example

As it can be seen, the line (acquired from literature) passes through the Peng-Robinson Equation of State plot three times. This is because the Peng-Robinson Equation of State has three roots. The largest root is the value of Z_v; the smallest root is the value of Z_L; and the third root is discarded as having no physical meaning.

Task 6.1: Calculating the Volumes at the roots

Calculate the volumes at the roots of the Peng-Robinson Equation of State for the gas, CO_2, at the temperature 284.0K and pressure at 4.6 MPa.

The equation for the volume is:

$$V = \frac{RTZ}{P}$$

where:

- R is the ideal gas constant
- T is the temperature
- P is the pressure
- Z is the root of the Peng-Robinson Equation of State

The volume variable names should be

V_v: The volume at the largest root

V_L: The volume at the smallest root

V_none: The volume at the middle root

> ***Tip***: Use the variables from the previous task

In [40]:
```
V_v = None # < - Enter your code here, remove None
V_L = None # < - Enter your code here, remove None
V_none = None # < - Enter your code here, remove None

print(f'Vapour volume: {V_v},\nNo Physical Meaning volume: {V_none},\nLiquid volume: {V_L}')
```

```
Vapour volume: None,
No Physical Meaning volume: None,
Liquid volume: None
```

In [41]:
```
### Teacher Solution

V_v = Z_v * R * temperature / pressure
V_L = Z_l * R * temperature / pressure
V_none = Z_none * R * temperature / pressure

print(f'Vapour volume: {V_v},\nNo Physical Meaning volume: {V_none},\nLiquid volume: {V_L}')
```

```
Vapour volume: 313.29149174634205,
No Physical Meaning volume: 119.14450353313504,
Liquid volume: 54.199076632870906
```

Task 6.2: Plot the volumes at the roots on the Peng-Robinson Equation of State Plot of CO_2 at 284.0K and 4.6 MPa

Plot the volumes at the roots on the Peng-Robinson Equation of State Plot of CO_2 at 284.0K and 4.6 MPa.

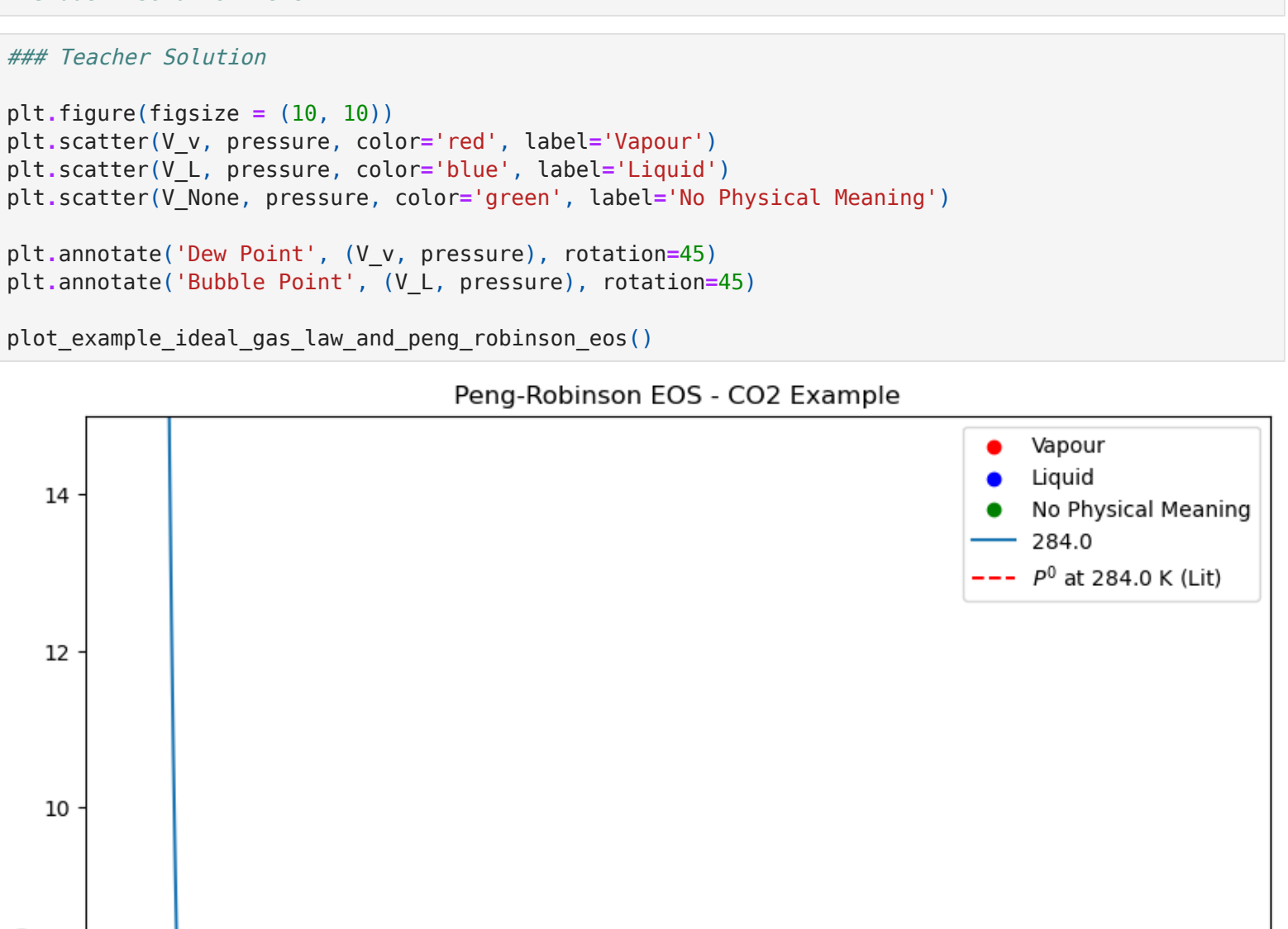
You will need to use the `plt.scatter()` function to plot the points. The function takes the following arguments: `plt.scatter(x, y, label)`. The x and y arguments are the x and y coordinates of the point and the label is what will be shown in the legend.

Further to this, you will need to annotate two of the points with either `Dew Point` or `Bubble Point`. You will need to use the `plt.annotate()` function to annotate the points. The function takes the following arguments: `plt.annotate(text, xy=(x, y), rotation=45)`. The text argument is the text to be displayed. The xy argument is the x and y coordinates of the point. The rotation argument is the rotation of the text. In this instance, it will be set to 45 degrees.

After you have plotted the points, you will need to use the the function `plot_example_ideal_gas_law_and_peng_robinson_eos()`, as can be seen in the example above of the plot.

In [42]:
```
# Student solution here
```

In [43]:
```
### Teacher Solution

plt.figure(figsize = (10, 10))
plt.scatter(V_v, pressure, color='red', label='Vapour')
plt.scatter(V_L, pressure, color='blue', label='Liquid')
plt.scatter(V_None, pressure, color='green', label='No Physical Meaning')

plt.annotate('Dew Point', (V_v, pressure), rotation=45)
plt.annotate('Bubble Point', (V_L, pressure), rotation=45)

plot_example_ideal_gas_law_and_peng_robinson_eos()
```



Task 7: Peng-Robinson Equation of State - Mixing Rules & Fugacity Coefficients

In the previous tasks we have only been looking at the Peng-Robinson Equation of State for a single component. However, in reality, we are often dealing with mixtures of components. In this task, we will look at how to use the Peng-Robinson Equation of State to predict the fugacity coefficients of a mixture of components.

The fugacity coefficient is defined as the ratio of fugacity to pressure [1]. For gases at low pressures (where the ideal gas law is a good approximation), fugacity is roughly equal to pressure. Thus, for an ideal gas, the ratio between fugacity and pressure (the fugacity coefficient) is equal to 1 [1]. This ratio can be thought of as 'how closely' a real gas behaves like an ideal gas [1].

Task 7.1: Peng-Robinson Equation of State - Mixing Rules

The next cell provides you with the variables required and the chemical data for components in a DataFrame. The DataFrame is called `chemical_data_df`

The chemical data contains the following columns:

- `Element`: The element name
- `Tc(K)`: The critical temperature of the element
- `Pc(MPa)`: The critical pressure of the element
- `Vc(cm3/mol)`: The critical volume of the element
- `w`: The acentric factor of the element

Below is the element names will be printed, as well as the number of rows and names of the columns.

In [44]:
```
# Shift + Enter to run the code
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

R = 8.314 # J/(mol*K)
T = 233.2 # K
P = 0.1 # MPa

chemical_data_df = pd.read_csv("src/Critical_Constants_and_Acentric_Factors.csv")

print(chemical_data_df.columns)
print('------------------')
print('Number of rows: ', chemical_data_df.shape[0])
print('Number of columns: ', chemical_data_df.shape[1])
print('------------------')
print('List of elements: ', chemical_data_df['Element'].unique())
```

```
Index(['Element', 'Tc_K', 'Pc_MPa', 'Vc_cm3_mol', 'w'], dtype='object')
------------------
Number of rows:  76
Number of columns:  5
------------------
List of elements:  ['Argon' 'Bromine' 'Chlorine' 'Fluorine' 'Helium-4' 'Hydrogen' 'Iodine'
 'Krypton' 'Neon' 'Nitrogen' 'Oxygen' 'Xenon' 'Acetylene' 'Benzene'
 'n-Butane' '1-Butene' 'Cyclobutane' 'Cyclohexane' 'Cyclopropane' 'Ethane'
 'Ethylene' 'n-Heptane' 'n-Hexane' 'Isobutane' 'Isobutylene' 'Isopentane'
 'Methane' 'Naphthalene' 'n-Octane' 'n-Pentane' 'Propadiene' 'Propane'
 'Propylene' 'Toluene' 'm-Xylene' 'o-Xylene' 'p-Xylene' 'Ammonia'
 'Carbon dioxide' 'Carbon disulfide' 'Carbon monoxide'
 'Carbon tetrachloride' 'Carbon tetrafluoride' 'Chloroform' 'Hydrazine'
 'Hydrogen chloride' 'Hydrogen fluoride' 'Hydrogen sulfide' 'Nitric oxide'
 'Nitrous oxide' 'Sulfur dioxide' 'Sulfur trioxide' 'Water' 'Acetaldehyde'
 'Acetic acid' 'Acetone' 'Acetonitrile' 'Aniline' 'n-Butanol'
 'Chlorobenzene' 'Dichlorodifluoromethane (Freon 12)' 'Diethyl ether'
 'Dimethyl ether' 'Ethanol' 'Ethylene oxide' 'Isobutanol'
 'Isopropyl alcohol' 'Methanol' 'Methyl chloride' 'Methyl ethyl ketone'
 'Phenol' '1-Propanol' 'Pyridine' 'Trichlorotrifluoroethane (Freon 113)'
 'Trichlorofluoromethane (Freon 11)' 'Trimethylamine']
```

Task 7.1.1: Peng-Robinson Equation of State - Mixing Rules - Binary Mixture

You have been provided with the chemical data for two components, Methane and Propane. The chemical data is stored in the DataFrame `chemical_data_df`.

The mol fractions of the components are:

- x_{CH_4} = 0.4
- $x_{C_3H_8}$ = 0.6

You will need to do the following:

1. Create a table of the chemical data for the two components, Methane_2 and Propane_2. The table should be called `chemical_data_df_binary_mixture`. It should have the following columns:
   - `Methane`
   - `Propane`

   It then should have the following index:
   - `mol_frac`
   - `Tc_K`
   - `Pc_MPa`
   - `w`
   - `ai`
   - `bi`
   - `ki`
2. Fill in the table with the correct values. The values for the `mol_frac` column should be the mol fractions of the components. The values for the `Tc_K` column should be the critical temperature of the components. The values for the `Pc_MPa` column should be the critical pressure of the components. The values for the `w` column should be the acentric factor of the components. The values for the `ai` column should be the attraction parameter of the component. The values for the `bi` column should be the covolume for the component. The values for the `ki` column should be the binary interaction of component.

> *Tip*: Creating an empty table (or rather, DataFrame) can be done using the `pd.DataFrame()` function. The function takes the following arguments: `pd.DataFrame(columns, index)`. The index argument is the index of the DataFrame and the columns argument is the columns of the DataFrame. These both accept only lists - `[]`.

> *Tip*: You will need to use the `chemical_data_df` DataFrame to get the values for the `Tc_K`, `Pc_MPa` and `w` columns. You will need to use the `ai()` function to get the values for the `ai` column. You will need to use the `bi()` function to get the values for the `bi` column. You will need to use the `ki()` function to get the values for the `ki` column.

> *Tip*: Access rows and setting them will require the use of the `.loc[]` function. The function takes the following arguments: `.loc[column]`. The column argument is the column to be accessed. The function will return the row of the column. To set the row of the column, you will need to use the following syntax: `.loc[column] = row`. The column argument is the column to be accessed. The row argument is the row to be set. Ensure the column is a string and the row is a list.

In [45]:
```
# Create the DataFrame
%%script false --no-raise-error
molecules = []
index_names = []
chemical_data_df_binary_mixture = pd.DataFrame(index=None, columns=None)
```

```
UsageError: Line magic function `%%script` not found.
```

In [46]:
```
### Teacher Solution

molecules = ['Methane', 'Propane']
index_names = ['mol_frac', 'Tc_K', 'Pc_MPa', 'w', 'ai', 'bi', 'ki']
chemical_data_df_binary_mixture = pd.DataFrame(index=index_names, columns=molecules)
print(chemical_data_df_binary_mixture)
```

```
         Methane Propane
mol_frac     NaN     NaN
Tc_K         NaN     NaN
Pc_MPa       NaN     NaN
w            NaN     NaN
ai           NaN     NaN
bi           NaN     NaN
ki           NaN     NaN
```

In [47]:
```
# Create the list of mole fractions and set it in the table
%%script false --no-raise-error
mol_frac = None # < - Enter your code here, remove None
chemical_data_df_binary_mixture.loc[''] = None # < - Enter your code here, remove None
print(chemical_data_df_binary_mixture)
```

```
UsageError: Line magic function `%%script` not found.
```

In [48]:
```
### Teacher Solution
mol_frac = [0.4, 0.6]
chemical_data_df_binary_mixture.loc['mol_frac'] = mol_frac
print(chemical_data_df_binary_mixture)
```

```
         Methane Propane
mol_frac     0.4     0.6
Tc_K         NaN     NaN
Pc_MPa       NaN     NaN
w            NaN     NaN
ai           NaN     NaN
bi           NaN     NaN
ki           NaN     NaN
```

In [49]:
```
from src.thermo import get_chemical_values
#?get_chemical_values # Uncomment this line to get help on the function

# Get the values for Tc_K - You will need to run the get_chemical_values function. Remember to use 

chemical_data_df_binary_mixture.loc['Tc_K'] = None # < - Enter your code here, remove None

# Get the values for Pc_MPa - You will need to run the get_chemical_values function. Remember to us
chemical_data_df_binary_mixture.loc['Pc_MPa'] = None # < - Enter your code here, remove None

# Get the values for w - You will need to run the get_chemical_values function. Remember to use ?ge
chemical_data_df_binary_mixture.loc['w'] = None # < - Enter your code here, remove None

print(chemical_data_df_binary_mixture)
```

```
         Methane Propane
mol_frac     0.4     0.6
Tc_K        None    None
Pc_MPa      None    None
w           None    None
ai           NaN     NaN
bi           NaN     NaN
ki           NaN     NaN
```

In [50]:
```
### Teacher Solution
chemical_data_df_binary_mixture.loc['Tc_K'] = get_chemical_values(chemical_data_df, molecules, 'Tc_
chemical_data_df_binary_mixture.loc['Pc_MPa'] = get_chemical_values(chemical_data_df, molecules, 'P
chemical_data_df_binary_mixture.loc['w'] = get_chemical_values(chemical_data_df, molecules, 'w')
chemical_data_df_binary_mixture
```

Out[50]:

	Methane	Propane
mol_frac	0.4	0.6
Tc_K	190.4	369.8
Pc_MPa	4.6	4.25
w	0.011	0.153
ai	NaN	NaN
bi	NaN	NaN
ki	NaN	NaN

So, now you will need to finish the filling in the last three columns of the table. The values for the `ai` column should be the attraction parameter of the component. The values for the `bi` column should be the covolume for the component. The values for the `ki` column should be the binary interaction of component.

The equation for the covolume parameter, `bi`, is:

$$b_i = 0.07780\frac{RT_c}{P_c}$$

where:

- R is the gas constant
- T_c is the critical temperature
- P_c is the critical pressure

The equation for the binary interaction parameter, `ki`, is:

$$k_i = 0.37464 + 1.54226\omega_i - 0.26992\omega_i^2$$

where:

- ω_i is the acentric factor

The equation for the attraction parameter, `ai`, is:

$$a_i = 0.45724\frac{R^2T_c^2}{P_c}\left(1 + \kappa_i(1 - \sqrt{T/T_c})\right)^2$$

where:

- R is the gas constant
- T_c is the critical temperature
- P_c is the critical pressure
- κ_i is the binary interaction parameters
- T is the temperature

For this section, you will need to use write the code to calculate `bi` and `ki` and then use the function `ai()`

> *Hint*: When you are setting the values for `chemical_data_df_binary_mixture.loc['{COLUMNNAME}']` you will need to use the `.loc[]` function in place of the variables required in the formulas. For example, for T_c, you will need to use `chemical_data_df.loc['Tc_K']` in place of T_c.

In [51]:
```
# Shift + Enter to run the code
from src.thermo import calculate_ai
```

In [52]:
```
# Calculate ki and bi AND THEN ai using the function `calculate_ai(chemical_df, molecules, temperat

chemical_data_df_binary_mixture.loc['ki'] = None # < - Enter your code here, remove None
chemical_data_df_binary_mixture.loc['bi'] = None # < - Enter your code here, remove None
chemical_data_df_binary_mixture.loc['ai'] = None # < - Enter your code here, remove None
chemical_data_df_binary_mixture
```

Out[52]:

	Methane	Propane
mol_frac	0.4	0.6
Tc_K	190.4	369.8
Pc_MPa	4.6	4.25
w	0.011	0.153
ai	None	None
bi	None	None
ki	None	None

In [53]:
```
### Teacher Solution

chemical_data_df_binary_mixture.loc['ki'] = 0.37464 + 1.54226 * chemical_data_df_binary_mixture.loc
chemical_data_df_binary_mixture.loc['bi'] = 0.07780 * R * chemical_data_df_binary_mixture.loc['Tc_K
chemical_data_df_binary_mixture.loc['ai'] = calculate_ai(chemical_data_df_binary_mixture, molecules
```

In [54]:
```
chemical_data_df_binary_mixture
```

Out[54]:

	Methane	Propane
mol_frac	0.4	0.6
Tc_K	190.4	369.8
Pc_MPa	4.6	4.25
w	0.011	0.153
ai	228729.152635	1285925.517754
bi	26.773104	56.28175
ki	0.391572	0.604287

Task 7.1.2: Peng-Robinson Equation of State - Mixing Rules - Interaction Parameters between components

Interaction parameters between components is usually acquired from literature. For this task, there is a DataFrame that has extracted the interaction parameters from literature. The DataFrame is called `interaction_parameters_df`. The DataFrame has the following columns:

- `component_1`: The name of the first component
- `component_2`: The name of the second componnet
- `k12`: The binary interaction parameter between the first and second component

You will need to create a table similar to this:

chemical_data_interaction_df

i, j	Methane	Ethane
Methane	0	0.00340
Ethane	0.0340	0

For this, you will need to create an empty DataFrame called `chemical_data_interaction_df`. The DataFrame should have your components as the columns **AND** the index.

Then, you will need to fill in the DataFrame with either 0 when the component is the same for both column and index, or the value from the `interaction_parameters_df` when the component is different for both column and index.

> *Tip*: Filtering a DataFrame will require .isin() function. An example of this is `filtered_df = df[(df['component_1'].str.upper().isin(filter_components)) & (df['component_2'].str.upper().isin(filter_components))]`.

> *Tip*: Once you have filtered the DataFrame, see what the values are and then you can use the `.loc['{COLUMNNAME}','{INDEXNAME}']` function to set the values

In [55]:
```
# Shift + Enter to run the code
import pandas as pd
interaction_parameter_df = pd.read_excel("src/interaction_parameters_df.xlsx")
interaction_parameter_df
```

Out[55]:

	#	component_1	component_2	k12
0	1	benzene	heptane	0.0011
1	2	carbon dioxide	benzene	0.0774
2	3	carbon dioxide	decane	0.1141
3	4	carbon dioxide	ethane	0.1322
4	5	carbon dioxide	heptane	0.1
...	...	...	...	...
56	57	nitrogen	propane	0.0852
57	58	nitrogen	toluene	0.20142
58	59	pentane	toluene	0.00845
59	60	propane	i-butane	−0.0078
60	61	propane	i-pentane	0.0111

61 rows × 4 columns

In [56]:
```
### Student Solution - 1

chemical_data_interaction_df = pd.DataFrame(index=None, columns=None) # < - Enter your code here, r

# filtered_df = df[(df['component_1'].str.upper().isin(filter_components)) & (df['component_2'].str
filtered_df = None
print(filtered_df)
```

```
None
```

In [57]:
```
### Student Solution - 2

chemical_data_interaction_df.loc['',''] = None # < - Enter your code here, remove None
chemical_data_interaction_df.loc['',''] = None # < - Enter your code here, remove None
chemical_data_interaction_df.loc['',''] = None # < - Enter your code here, remove None
chemical_data_interaction_df.loc['',''] = None # < - Enter your code here, remove None

print(chemical_data_interaction_df)
```

```
 NaN
```

In [58]:
```
### Teacher Solution - 1

chemical_data_interaction_df = pd.DataFrame(index=molecules, columns=molecules)
chemical_data_interaction_df

filtered_df = interaction_parameter_df[(interaction_parameter_df['component_1'].str.title().isin(mo
print(filtered_df)
```

```
     # component_1 component_2    k12
44  45     methane     propane  0.014
```

In [59]:
```
### Teacher Solution - 2

chemical_data_interaction_df.loc['Methane','Propane'] = 0.014 # < - Enter your code here, remove No
chemical_data_interaction_df.loc['Propane','Methane'] = 0.014 # < - Enter your code here, remove No
chemical_data_interaction_df.loc['Methane','Methane'] = 0 # < - Enter your code here, remove None
chemical_data_interaction_df.loc['Propane','Propane'] = 0 # < - Enter your code here, remove None

print(chemical_data_interaction_df)
```

```
        Methane Propane
Methane       0   0.014
Propane   0.014       0
```

Task 7.2: Peng-Robinson Equation of State - Mixing Rules - Creating aij table

In this section we will now need to create the aij table. The aij table is a table that contains the attraction parameters between components. The table will be called `aij_df`.

Remember, the equation for attraction parameter, `aij` is:

$$a_{ij} = \sqrt{a_i a_j}(1 - \kappa_{ij})$$

where:

- a_i is the attraction parameter of the first component
- a_j is the attraction parameter of the second component
- κ_{ij} is the binary interaction parameter between the two components

> *Tip*: You will need to use the function from the module `thermo` called `calculate_df_aij()` to calculate the aij table. The function takes the following arguments: `calculate_df_aij(molecules, chemical_data_df_binary_mixture, chemical_data_interaction_df, df_aij)`.

You table will be similar to the interaction table set up, where the columns and the index are the components.

In [60]:
```
# Shift + Enter to run the code
from src.thermo import calculate_df_aij
```

In [61]:
```
### Student Solution

%%script false --no-raise-error

df_aij = pd.DataFrame(index=None, columns=None) # < - Enter your code here, remove None

df_aij = calculate_df_aij()
```

```
UsageError: Line magic function `%%script` not found.
```

In [62]:
```
### Teacher Solution

df_aij = pd.DataFrame(index=molecules, columns=molecules)
df_aij = calculate_df_aij(molecules, chemical_data_df_binary_mixture, chemical_data_interaction_df,
df_aij
```

Out[62]:

	Methane	Propane
Methane	228729.152635	534743.584282
Propane	534743.584282	1285925.517754

Task 7.2.1: Peng-Robinson Equation of State - Mixing Rules - Finding the value of `a`

Now that we have our `aij` table, we will need to find the value of `a`. The equation for `a` is:

$$a = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j a_{ij}$$

where:

- x_i is the mole fraction of the first component
- x_j is the mole fraction of the second component
- a_{ij} is the attraction parameter between the two components

Create a DataFrame called `a_mol_frac_df` that has columns and indexes again defined by the molecules list.

> ***Python**: Use vectorization to calculate the values for the `a_mol_frac_df` DataFrame. To do this, you will need to lock onto the mol fractions of the components and then multiply them by the aij table. `.multiply()` is a good function to use. Make sure you choose the right axis! You may have to chain the `.multiply()` function twice.* ALTERNATIVELY* you can use two for loops to calculate the values for the DataFrame.

Once you have completed this table, you will need to sum all the values in the table. This will give you the value of `a`

In [63]:
```
### Student Solution

# Get the mol fractions in a list

# Use the multiply function, with the axis=0 and then axis=1

# Get the sum of the values in the dataframe - use the sum function, twice!
```

In [64]:
```
### Teacher Solution

# Get the mol fractions in a list
mol_frac = chemical_data_df_binary_mixture.loc['mol_frac'].tolist()

# Use the multiply function, with the axis=0 and then axis=1
df_aij_mol_frac = df_aij.multiply(mol_frac, axis=0).multiply(mol_frac, axis=1)
print(df_aij_mol_frac)
# Get the sum of the values in the dataframe
a = df_aij_mol_frac.sum().sum()
print(a)
```

```
               Methane        Propane
Methane   36596.664422  128338.460228
Propane  128338.460228  462933.186391
756206.7712684579
```

Task 7.2.1: Peng-Robinson Equation of State - Mixing Rules - Finding the value of `b`

Finally, we will need to find the value of `b`. The equation for `b` is:

$$b = \sum_{i=1}^{n} x_i b_i$$

where:

- x_i is the mole fraction of the component i
- b_i is the covolume of the component i

> *Tip*: You can use `.loc[]` to access the relevant indexes to multiply. Then after the multiplication, you can use `.sum()` to sum all the values.

In [65]:
```
### Student Solution
%%script false --no-raise-error
b = (df.loc[''] * df.loc['']).sum() # <- adjust the relevant section of the code to get `b`
print(b)
```

```
UsageError: Line magic function `%%script` not found.
```

In [66]:
```
### Teacher Solution
b = (chemical_data_df_binary_mixture.loc['mol_frac'] * chemical_data_df_binary_mixture.loc['bi']).s
print(b)
```

```
44.47829180448082
```

Task 7.3: Peng-Robinson Equation of State - Mixing Rules - Finding the roots of the cubic equation

We have all the data we need now to find the roots of the cubic equation. Recall, the cubic equation is:

$$Z^3 + (1 - B)Z^2 + (A - 3B^2 - 2B)Z - (AB - B^2 - B^3) = 0$$

where:

- Z is the compressibility factor
- A is the value calculated from `a`
- B is the value calculated from `b`

You will need to find:

1. The value of `A`
2. The value of `B`
3. Then the values of each coefficient:
   - Z^3 -> 1
   - Z^2 -> $(1 - B)$
   - Z^1 -> $(A - 3B^2 - 2B)$
   - Z^0 -> $(-(AB - B^2 - B^3))$
4. Use the CubicEquationSolver module to find the roots using the function `solve(a, b, c, d)`

In [67]:
```
### Student Solution
%%script false --no-raise-error
# Find the value of A
A = None # <- Enter your code here, remove None

# Find the value of B
B = None # <- Enter your code here, remove None

# Find the coefficients
Z_3 = None # <- Enter your code here, remove None
Z_2 = None # <- Enter your code here, remove None
Z_1 = None # <- Enter your code here, remove None
Z_0 = None # <- Enter your code here, remove None

# Find the roots of the cubic equation
import CubicEquationSolver as ces
Z_roots = ces.solve(Z_3, Z_2, Z_1, Z_0)
print(Z_roots)
```

```
UsageError: Line magic function `%%script` not found.
```

In [68]:
```
### Teacher Solution

# Find the value of A
A = a * P / (R * T)**2

# Find the value of B
B = b * P / (R * T)

# Find the coefficients
Z_3 = 1
Z_2 = -1 - B
Z_1 = A - 3 * B**2 - 2 * B
Z_0 = -A * B + B**2 + B**3

# Find the roots of the cubic equation
import CubicEquationSolver as ces
Z_roots = ces.solve(Z_3, Z_2, Z_1, Z_0)
print(Z_roots)
```

```
[0.98661253 0.00336329 0.01231827]
```

Task 7.4: Peng-Robinson Equation of State - Mixing Rules - Fugacity Coefficient

The fugacity coefficient is a dimensionless quantity that relates the fugacity of a real gas to its pressure. The real gas pressure and fugacity are related through the dimensionless fugacity coefficient φ [1]. For an ideal gas, fugacity and pressure are equal and so φ = 1. Taken at the same temperature and pressure, the difference between the molar Gibbs free energies of a real gas and the corresponding ideal gas is equal to RT $ln(\varphi)$ [1]. Fugacities are determined experimentally or estimated from various models such as a Van der Waals gas that are closer to reality than an ideal gas [1].

Fugacity Coefficient

$$\ln\phi_k = \frac{b_k}{b}(Z-1) - \ln(Z-B) - \frac{A}{2\sqrt{2}B}\left(\frac{2\sum_{i=1}^{n} x_i a_{ik}}{a} - \frac{b_k}{b}\right)\ln\left(\frac{Z + 2.414B}{Z - 0.414B}\right)$$

where ϕ_k is the fugacity coefficient for component k;
Z is the compressibility factor;
a is the Peng-Robinson attraction parameter for the mixture;
a_{ik} is the cross Peng-Robinson attraction parameter for components i and k;
b is the Peng-Robinson covolume for the mixture;
and b_k is the Peng-Robinson covolume for component k;

$$A = \frac{aP}{R^2T^2} \qquad B = \frac{bP}{RT}$$

where P is the pressure (Pa);
R is the gas constant (8.314 J mol^{-1} K^{-1});
T is the absolute temperature (K);
a is the Peng-Robinson attraction parameter for the mixture;
and b is the Peng-Robinson covolume for the mixture.

We have done a lot of work so far, and as seen by the equation above, we have a lot of data we need to work with. However, for this exercise, we will use a script called `preos` that has been developed to calculate the fugacity coefficient and other properties.

Source: https://github.com/CorySimon/PREOS

Provided is an example of loading a molecule into the `preos` module. You will need to do this for each molecule in the `molecules` list. The class `Molecule` takes the following arguments: `Molecule(name, temperature, pressure, omega)`. The `name` is the name of the molecule, `temperature` is the critical temperature in Kelvin, `pressure` is the critical pressure in bar , and `omega` is the acentric of the molecule.

```
xe = preos.Molecule("Xe", 16.59 + 273.15, 58.42, 0.0)
xe.print_params()
```

Make sure to use the chemical_data_df_binary_mixture DataFrame you created!

In [69]:
```
# Shift + Enter to run the code
import src.preos as preos
```

In [70]:
```
### Student Solution

# save component 1 and component 2 in variables using preos.Molecule
```

In [86]:
```
### Teacher Solution

methane = preos.Molecule('Methane', 190.4, 46, 0.011)
propane = preos.Molecule('Propane', 369.8, 42.5, 0.153)
```

Now, for the last part.

The `preos` module contains a function called `.preos_mixture()` that takes the following arguments:

- molecule_a
- molecule_b
- delta - the binary interaction parameter between the two molecules
- T - the temperature in Kelvin chosen
- P - the pressure in bar chosen
- x - the mole fraction as list [x_a, x_b]

For delta, make sure to use `interaction_parameter_df`

In [87]:
```
### Student Solution
%% script false --no-raise-error
T = None # <- Enter your code here, remove None
P = None # <- Enter your code here, remove None
x = list() # <- Enter your code here, remove None
delta = None # <- Enter your code here, remove None

mixture = preos.preos_mixture()
```

```
UsageError: Line magic function `%%` not found.
```

In [88]:
```
### Teacher Solution

T = 233.2
P = 0.1/0.1
x = [0.4, 0.6]
delta =0.0114
mixture = preos.preos_mixture(methane, propane,delta,  T, P, x )
```

```
PREOS calculation at
                 T = 233.20 K
                 P, total = 1.00 bar
         Density: 52.526575 mol/m3
         Compressibility factor : 0.981934
Component 0, Methane:
         Fugacity coefficient: 0.999675
         Fugacity: 0.399870 bar
Component 1, Propane:
         Fugacity coefficient: 0.970735
         Fugacity: 0.582441 bar
```