# LAPORAN UJI COBA PRAKTIKUM - KECERDASAN KOMPUTASIONAL \n

### Tujuan Praktikum : Menguji Coba Metode Fuzzy Logic dengan menggunakan Library `scikit-fuzzy` python \n

**Nama : Abdan Hafidz**

**NRP : 5054231021**

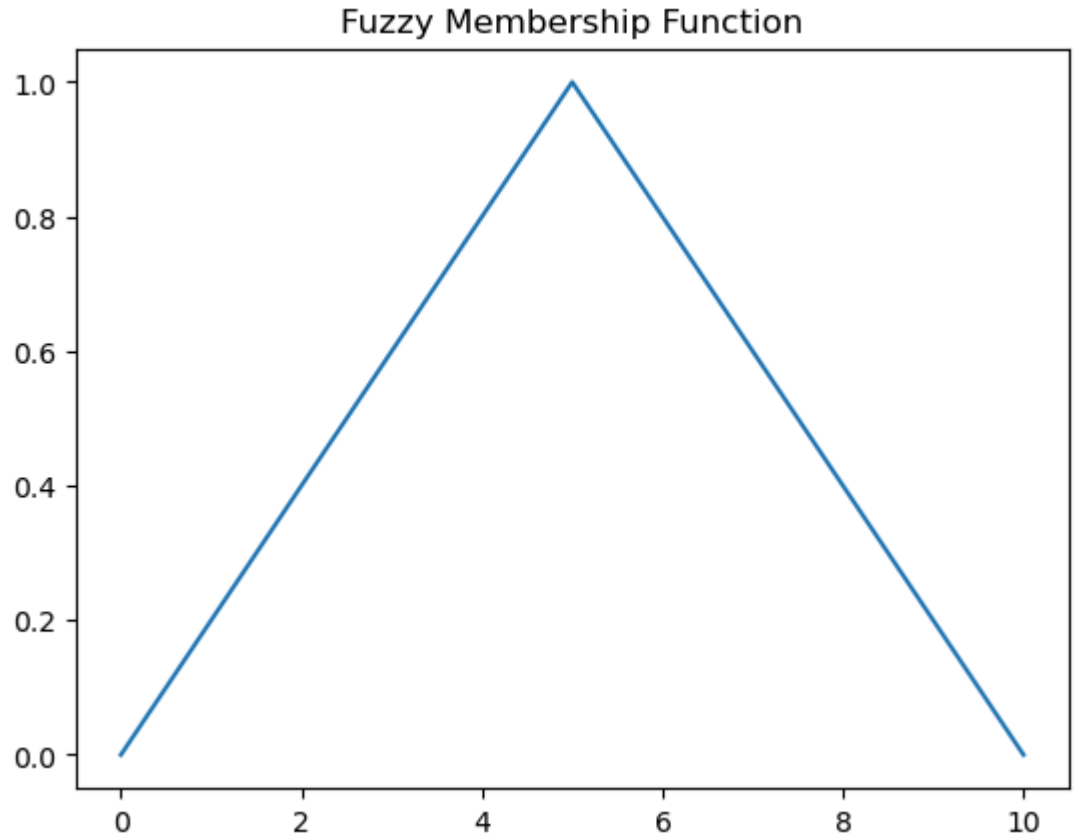
## Getting Started

This code aims to visualize a triangular *fuzzy membership function* using the `scikit-fuzzy` library. First, an array `x` is created with values ranging from 0 to 10 using `np.arange()`. Then, a triangular fuzzy membership function is defined with `fuzz.trimf()`, where the start point (0), midpoint (5), and endpoint (10) are specified as parameters. The resulting plot illustrates the relationship between the input values of `x` and their corresponding fuzzy membership degrees, providing a visual representation of how membership functions work in a fuzzy logic system.

In [2]:

```
import numpy as np
import skfuzzy as fuzz
from matplotlib import pyplot as plt

# [0, 10]
x = np.arange(11)
mfx = fuzz.trimf(x, [0, 5, 10])

print(x)
print(mfx)

plt.title("Fuzzy Membership Function")
plt.plot(x, mfx)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10]
[0.  0.2 0.4 0.6 0.8 1.  0.8 0.6 0.4 0.2 0. ]
```
Out[2]:
```
[<matplotlib.lines.Line2D at 0x1d80a849110>]
```




## Membership Function

In this script, multiple types of fuzzy membership functions are generated and plotted to compare their shapes and characteristics. The `np.arange()` function is used to create an array `x` ranging from 0 to 10 with a step of 0.25, representing the input values. Different membership functions are then created using `scikit-fuzzy`:

- **Triangular Membership Function** ( `trimf` ): Defined by three points (0, 5, 10), representing the base and peak of the triangle.
- **Trapezoidal Membership Function** ( `trapmf` ): Defined by four points (0, 2, 8, 10), giving it a trapezoidal shape.

- **Sigmoid Membership Function** ( `sigmf` ): Controlled by a center (5) and a width parameter (2), giving a smooth S-shaped curve.
- **S-function** ( `smf` ): A smoother function defined by a foot (1) and ceiling (9).
- **Z-function** ( `zmf` ): Complementary to the S-function, used to describe gradual downward slopes.
- **Pi-function** ( `pimf` ): A smooth transition between two S-functions, defined by four parameters (0, 4, 5, 10).
- **Gaussian Function** ( `gaussmf` ): A symmetric bell curve controlled by mean (5) and standard deviation (1.25).
- **Generalized Bell-Shaped Function** ( `gbellmf` ): Similar to the Gaussian but controlled by a width (2), slope (4), and center (5).

Each of these membership functions is plotted in three subplots, providing a comparison between different fuzzy logic membership models. The plot gives insight into how different shapes of membership functions can be used for modeling uncertainty and gradual transitions in fuzzy systems.

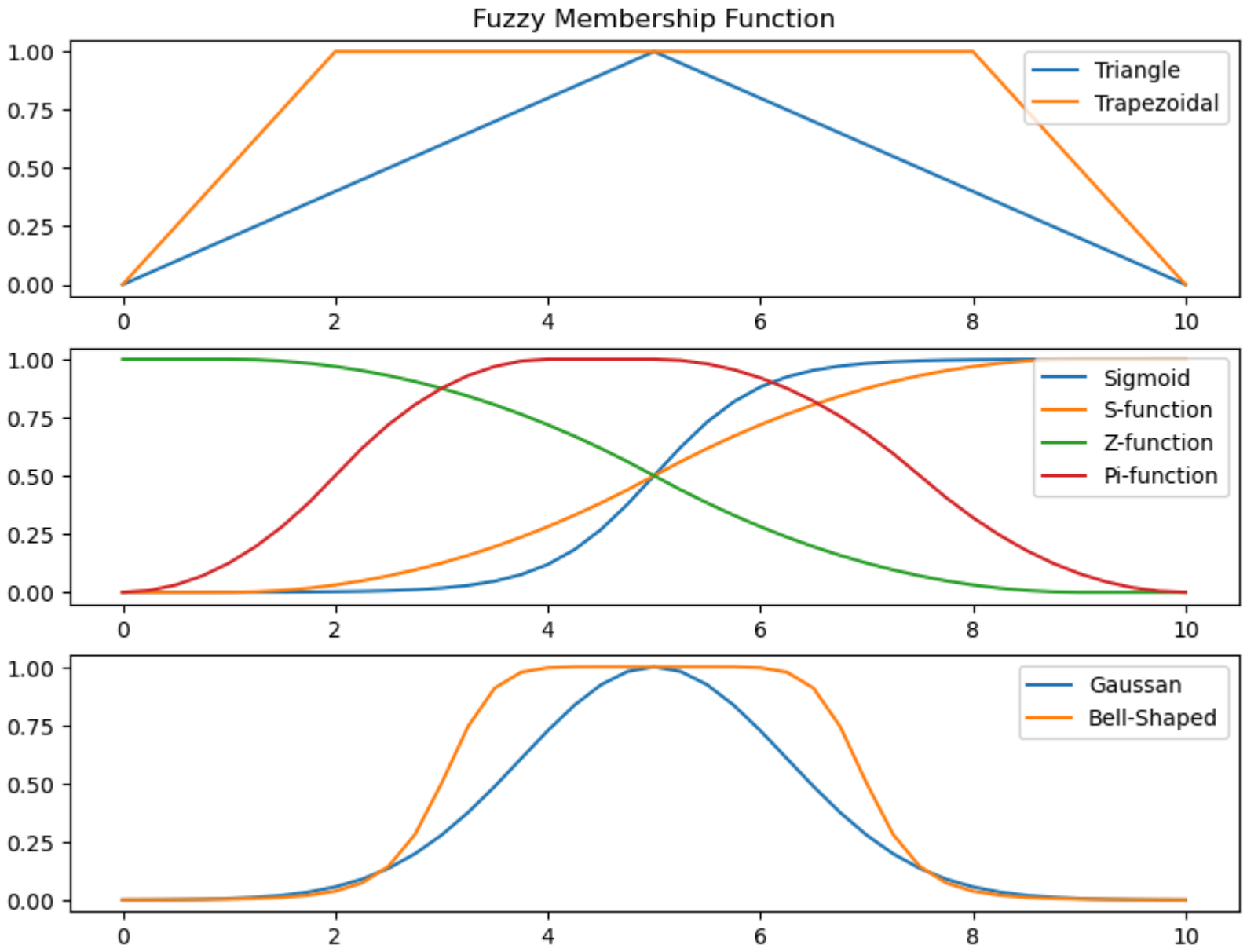
In [3]:

```
# [0, 10]
start = 0
stop = 10 + 0.001
step = 0.25
x = np.arange(start, stop, step)
print(x)

# Triangular membership function
trimf = fuzz.trimf(x, [0, 5, 10])

# Trapezoidal membership function
trapmf = fuzz.trapmf(x, [0, 2, 8, 10])

# Sigmoid membership function
center = 5.0
width_control = 2.0
sigmf = fuzz.sigmf(x, center, width_control)
```

```
# S-function
foot = 1.0
ceiling = 9.0
smf = fuzz.smf(x, foot, ceiling)

# Z-function
zmf = fuzz.zmf(x, foot, ceiling)

# Pi-function
pimf = fuzz.pimf(x, 0.0, 4.0, 5.0, 10.0)

# Gaussian function
mean = 5.0
sigma = 1.25
gaussmf = fuzz.gaussmf(x, mean, sigma)

# Generalized Bell-Shaped function
width = 2.0
slope = 4.0
center = 5.0
gbellmf = fuzz.gbellmf(x, width, slope, center)

fig_scale = 1.5
plt.figure(figsize=(6.4 * fig_scale, 4.8 * fig_scale))

# 3 rows, 1 col, index from 1
plt.subplot(311)
plt.title("Fuzzy Membership Function")
```

```
plt.plot(x, trimf, label="Triangle")
plt.plot(x, trapmf, label="Trapezoidal")
plt.legend(loc="upper right")

plt.subplot(312)
plt.plot(x, sigmf, label="Sigmoid")
plt.plot(x, smf, label="S-function")
plt.plot(x, zmf, label="Z-function")
plt.plot(x, pimf, label="Pi-function")
plt.legend(loc="upper right")

plt.subplot(313)
plt.plot(x, gaussmf, label="Gaussan")
plt.plot(x, gbellmf, label="Bell-Shaped")
plt.legend(loc="upper right")
plt.show()
```

```
[ 0.    0.25  0.5   0.75  1.    1.25  1.5   1.75  2.    2.25  2.5   2.75
  3.    3.25  3.5   3.75  4.    4.25  4.5   4.75  5.    5.25  5.5   5.75
  6.    6.25  6.5   6.75  7.    7.25  7.5   7.75  8.    8.25  8.5   8.75
  9.    9.25  9.5   9.75 10.  ]
```


Fuzzy Membership Function
Triangle
Trapezoidal
Sigmoid
S-function
Z-function
Pi-function
Gaussan
Bell-Shaped
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10

## Fuzzy Logic

This code demonstrates the application of fuzzy logic operators on two membership functions, specifically the triangular and trapezoidal membership functions. The workflow is broken down as follows:

1. **Membership Functions Creation**:
   - A triangular membership function ( `trimf` ) is created over the range [0, 5] with a peak at 2.5, simulating a gradual rise and fall.
   - A trapezoidal membership function ( `trapmf` ) is defined over the range [4, 10] with flat sections between 6 and 8.
2. **Fuzzy Logical Operations**:
   - **Fuzzy NOT** ( `fuzz.fuzzy_not` ): The negation of the triangular and trapezoidal membership functions is computed, inverting their membership values.
   - **Fuzzy AND** ( `fuzz.fuzzy_and` ): The intersection of the two membership functions is calculated, producing values that represent the minimum of the two functions.
   - **Fuzzy OR** ( `fuzz.fuzzy_or` ): The union of the two membership functions is computed, representing the maximum values between the two functions.
3. **Visualization**:
   - The first plot shows the original triangular and trapezoidal membership functions.
   - The second plot visualizes the results of applying the Fuzzy NOT operator to both functions.
   - The third plot illustrates the results of the Fuzzy AND and Fuzzy OR operations between the two functions.

This script highlights how basic fuzzy logic operations such as NOT, AND, and OR can be applied to membership functions, providing a way to model logical relationships within fuzzy systems.

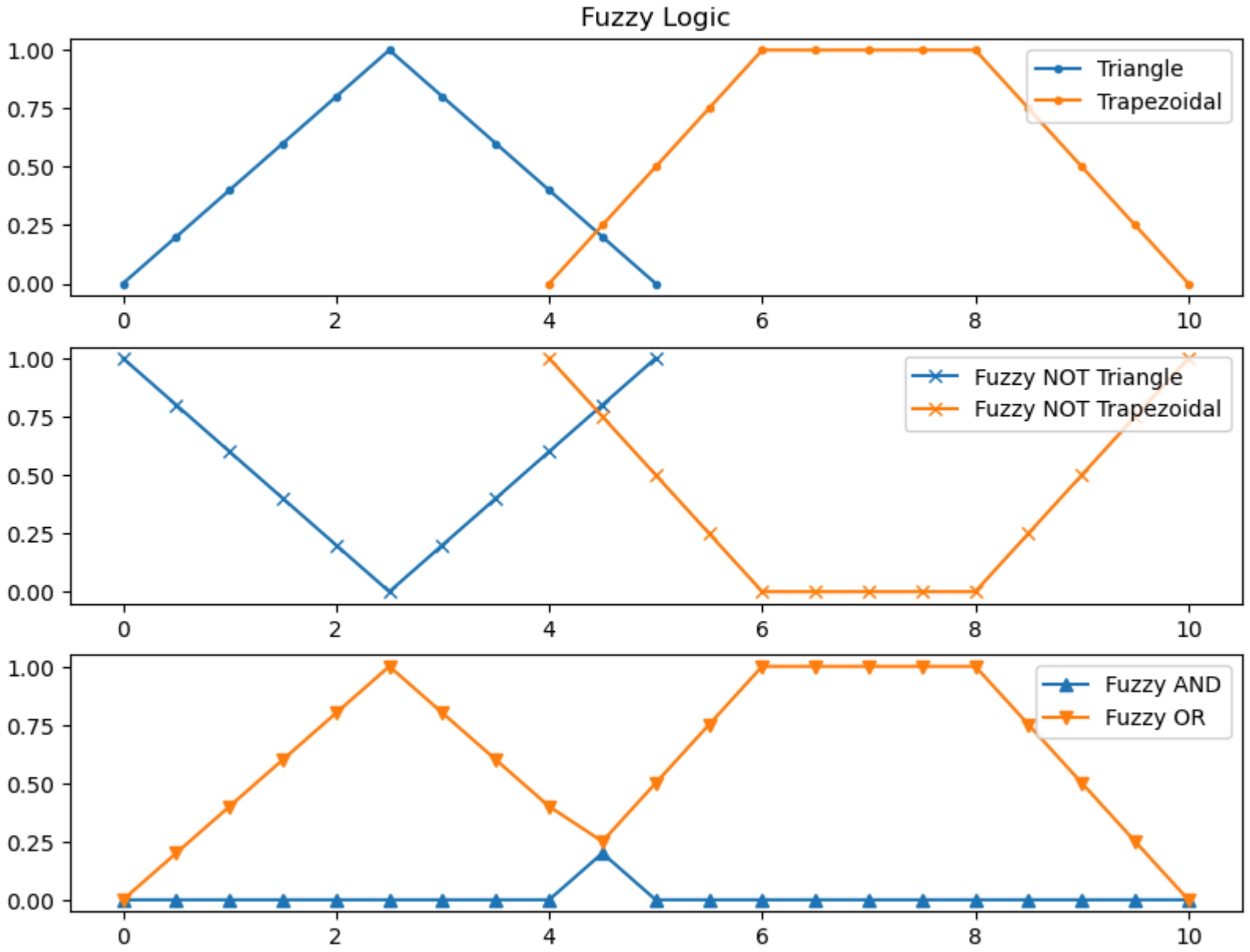
In [5]:

```
# [0, 10]
delta = 0.001
start = 0
stop = 10 + delta
step = 0.5
x = np.arange(start, stop + delta, step)
```

```
# Triangular membership function
x1 = np.arange(0, 5 + delta, step)
trimf = fuzz.trimf(x1, [0, 2.5, 5])

# Trapezoidal membership function
x2 = np.arange(4, 10 + delta, step)
trapmf = fuzz.trapmf(x2, [4, 6, 8, 10])

# fuzzy logic
tri_not = fuzz.fuzzy_not(trimf)
trap_not = fuzz.fuzzy_not(trapmf)
x3, tri_trap_and = fuzz.fuzzy_and(x1, trimf, x2, trapmf)
x3, tri_trap_or = fuzz.fuzzy_or(x1, trimf, x2, trapmf)
print(x2)
print(x3)

# Whole config
fig_scale = 1.5
plt.figure(figsize=(6.4 * fig_scale, 4.8 * fig_scale))
row = 3
col = 1

plt.subplot(row, col, 1)
plt.title("Fuzzy Logic")
plt.plot(x1, trimf, label="Triangle", marker=".")
plt.plot(x2, trapmf, label="Trapezoidal", marker=".")
plt.legend(loc="upper right")
```

```
plt.subplot(row, col, 2)
plt.plot(x1, tri_not, label="Fuzzy NOT Triangle", marker="x")
plt.plot(x2, trap_not, label="Fuzzy NOT Trapezoidal", marker="x")
plt.legend(loc="upper right")

plt.subplot(row, col, 3)
plt.plot(x3, tri_trap_and, label="Fuzzy AND", marker="^")
plt.plot(x3, tri_trap_or, label="Fuzzy OR", marker="v")
plt.legend(loc="upper right")
plt.show()
```

```
[ 4.   4.5  5.   5.5  6.   6.5  7.   7.5  8.   8.5  9.   9.5 10. ]
[ 0.   0.5  1.   1.5  2.   2.5  3.   3.5  4.   4.5  5.   5.5  6.   6.5
  7.   7.5  8.   8.5  9.   9.5 10. ]
```




## Defuzzyfication

This code applies defuzzification methods on fuzzy membership functions to extract crisp values. After generating triangular ( `trimf` ) and trapezoidal ( `trapmf` ) membership functions, fuzzy logic operations such as NOT, AND, and OR are applied. The result of the OR operation is then defuzzified using various methods:

- **Centroid**: Calculates the center of gravity of the membership function.
- **Bisector**: Splits the area under the curve into two equal parts.
- **MOM (Mean of Maximum)**: Averages the maximum membership values.
- **SOM (Smallest of Maximum)** and **LOM (Largest of Maximum)**: Return the smallest and largest values of the maximum membership, respectively.

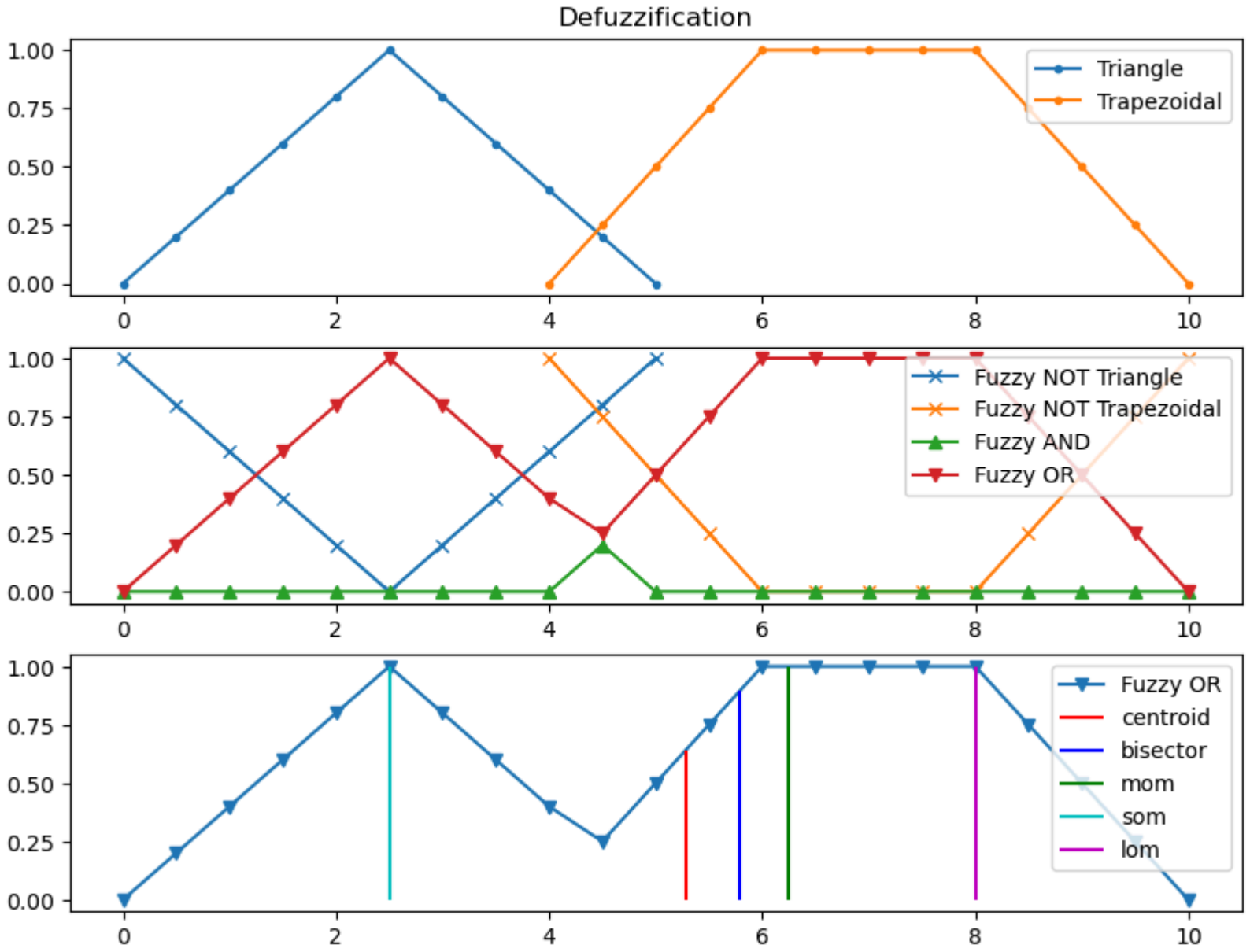
The final plot visualizes these defuzzified values as vertical lines on the graph, indicating the crisp results derived from the fuzzy set.

In [7]:

```
# [0, 10]
delta = 0.001
start = 0
stop = 10 + delta
step = 0.5
x = np.arange(start, stop + delta, step)

# Triangular membership function
x1 = np.arange(0, 5 + delta, step)
trimf = fuzz.trimf(x1, [0, 2.5, 5])

# Trapezoidal membership function
x2 = np.arange(4, 10 + delta, step)
trapmf = fuzz.trapmf(x2, [4, 6, 8, 10])

# fuzzy logic
tri_not = fuzz.fuzzy_not(trimf)
trap_not = fuzz.fuzzy_not(trapmf)
```

```
x3, tri_trap_and = fuzz.fuzzy_and(x1, trimf, x2, trapmf)
x3, tri_trap_or = fuzz.fuzzy_or(x1, trimf, x2, trapmf)

# Defuzzify
centroid_x = fuzz.defuzz(x3, tri_trap_or, "centroid")
centroid_y = fuzz.interp_membership(x3, tri_trap_or, centroid_x)
bisector_x = fuzz.defuzz(x3, tri_trap_or, "bisector")
bisector_y = fuzz.interp_membership(x3, tri_trap_or, bisector_x)
mom_x = fuzz.defuzz(x3, tri_trap_or, "mom")
mom_y = fuzz.interp_membership(x3, tri_trap_or, mom_x)
som_x = fuzz.defuzz(x3, tri_trap_or, "som")
som_y = fuzz.interp_membership(x3, tri_trap_or, som_x)
lom_x = fuzz.defuzz(x3, tri_trap_or, "lom")
lom_y = fuzz.interp_membership(x3, tri_trap_or, lom_x)

# Whole config
fig_scale = 1.5
plt.figure(figsize=(6.4 * fig_scale, 4.8 * fig_scale))
row = 3
col = 1

plt.subplot(row, col, 1)
plt.title("Defuzzification")
plt.plot(x1, trimf, label="Triangle", marker=".")
plt.plot(x2, trapmf, label="Trapezoidal", marker=".")
plt.legend(loc="upper right")

plt.subplot(row, col, 2)
```

```
plt.plot(x1, tri_not, label="Fuzzy NOT Triangle", marker="x")
plt.plot(x2, trap_not, label="Fuzzy NOT Trapezoidal", marker="x")
plt.plot(x3, tri_trap_and, label="Fuzzy AND", marker="^")
plt.plot(x3, tri_trap_or, label="Fuzzy OR", marker="v")
plt.legend(loc="upper right")

plt.subplot(row, col, 3)
plt.plot(x3, tri_trap_or, label="Fuzzy OR", marker="v")
plt.vlines(centroid_x, 0.0, centroid_y, label="centroid", color="r")
plt.vlines(bisector_x, 0.0, bisector_y, label="bisector", color="b")
plt.vlines(mom_x, 0.0, mom_y, label="mom", color="g")
plt.vlines(som_x, 0.0, som_y, label="som", color="c")
plt.vlines(lom_x, 0.0, lom_y, label="lom", color="m")
plt.legend(loc="upper right")
plt.show()
```




**SAMPLE PROBLEM : Temperature Control in a Greenhouse**

Problem Overview

This example models a system that controls the heating in a greenhouse based on two input factors:

- Humidity : How humid the environment inside the greenhouse is.
- Temperature : The temperature inside the greenhouse. The output will be the Heating Level, determining how much the heating system needs to operate to maintain optimal conditions for plant growth.

## Variables

### Input Variables:

- Humidity: Low, Medium, High.
- Temperature: Cold, Comfortable, Hot.
- Output Variable: ** Heating Level: Low, Moderate, High.

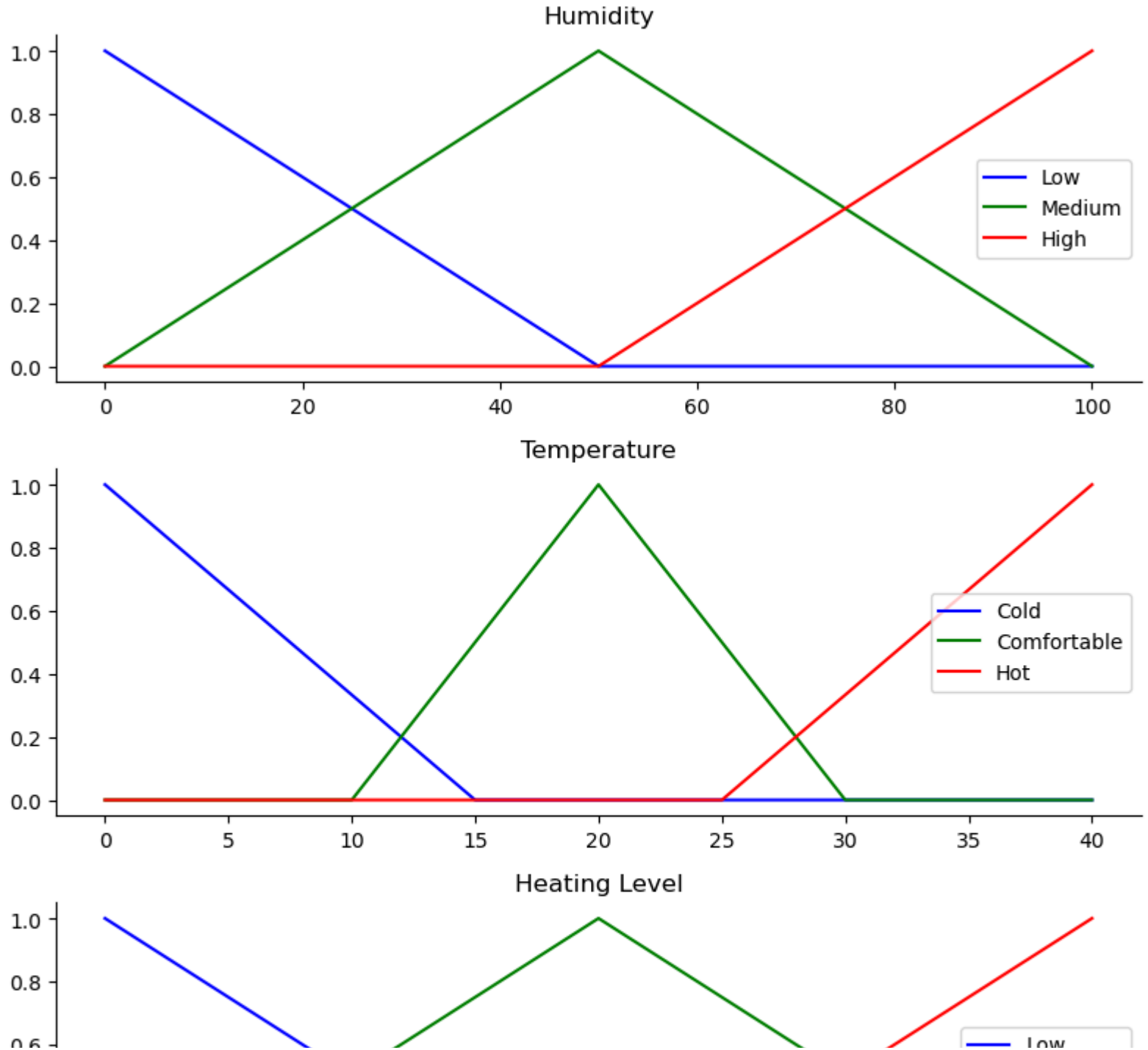
The fuzzy system uses triangular membership functions for the input variables and the output variable.

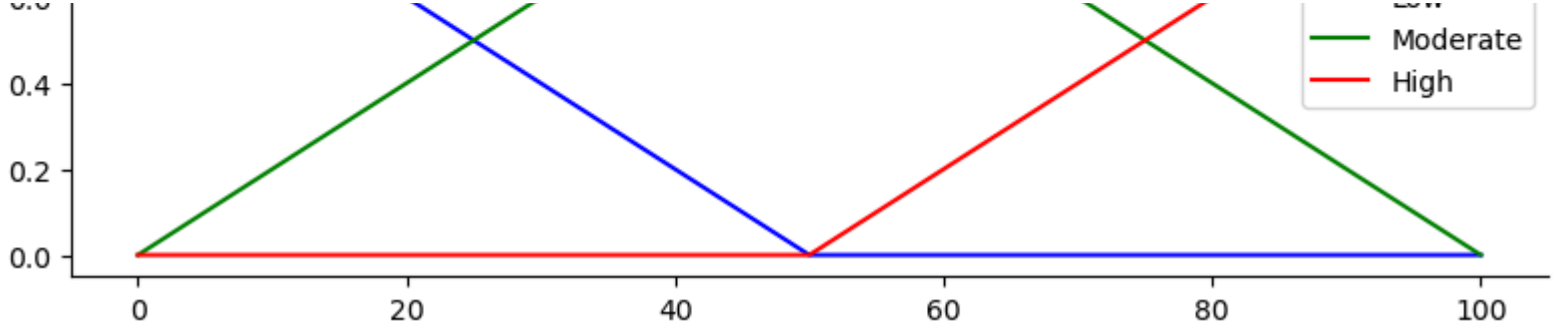
In [8]:

```
# Define universe variables
#   * Humidity on the range [0, 100] percentage
#   * Temperature on the range [0, 40] degrees Celsius
#   * Heating Level on the range [0, 100] percentage
x_humidity = np.arange(0, 101, 1)
x_temp = np.arange(0, 41, 1)
x_heating = np.arange(0, 101, 1)

# Generate fuzzy membership functions
humidity_lo = fuzz.trimf(x_humidity, [0, 0, 50])
humidity_md = fuzz.trimf(x_humidity, [0, 50, 100])
humidity_hi = fuzz.trimf(x_humidity, [50, 100, 100])

temp_cold = fuzz.trimf(x_temp, [0, 0, 15])
temp_comfy = fuzz.trimf(x_temp, [10, 20, 30])
```

```
temp_hot = fuzz.trimf(x_temp, [25, 40, 40])

heating_lo = fuzz.trimf(x_heating, [0, 0, 50])
heating_md = fuzz.trimf(x_heating, [0, 50, 100])
heating_hi = fuzz.trimf(x_heating, [50, 100, 100])

# Visualize the membership functions for input and output variables
fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(8, 9))

ax0.plot(x_humidity, humidity_lo, 'b', linewidth=1.5, label='Low')
ax0.plot(x_humidity, humidity_md, 'g', linewidth=1.5, label='Medium')
ax0.plot(x_humidity, humidity_hi, 'r', linewidth=1.5, label='High')
ax0.set_title('Humidity')
ax0.legend()

ax1.plot(x_temp, temp_cold, 'b', linewidth=1.5, label='Cold')
ax1.plot(x_temp, temp_comfy, 'g', linewidth=1.5, label='Comfortable')
ax1.plot(x_temp, temp_hot, 'r', linewidth=1.5, label='Hot')
ax1.set_title('Temperature')
ax1.legend()

ax2.plot(x_heating, heating_lo, 'b', linewidth=1.5, label='Low')
ax2.plot(x_heating, heating_md, 'g', linewidth=1.5, label='Moderate')
ax2.plot(x_heating, heating_hi, 'r', linewidth=1.5, label='High')
ax2.set_title('Heating Level')
ax2.legend()

for ax in (ax0, ax1, ax2):
```

```
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

plt.tight_layout()
plt.show()
```


Humidity
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
Low
Medium
High
Temperature
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
35
40
Cold
Comfortable
Hot
Heating Level
1.0
0.8
0.6
Low




## Fuzzy Rules

Now we define the rules that govern the behavior of the system. For this example, let's consider three rules:

1. If the humidity is low OR the temperature is cold, then the heating level will be high.
2. If the humidity is medium AND the temperature is comfortable, then the heating level will be moderate.
3. If the humidity is high OR the temperature is hot, then the heating level will be low.

### Rule Application

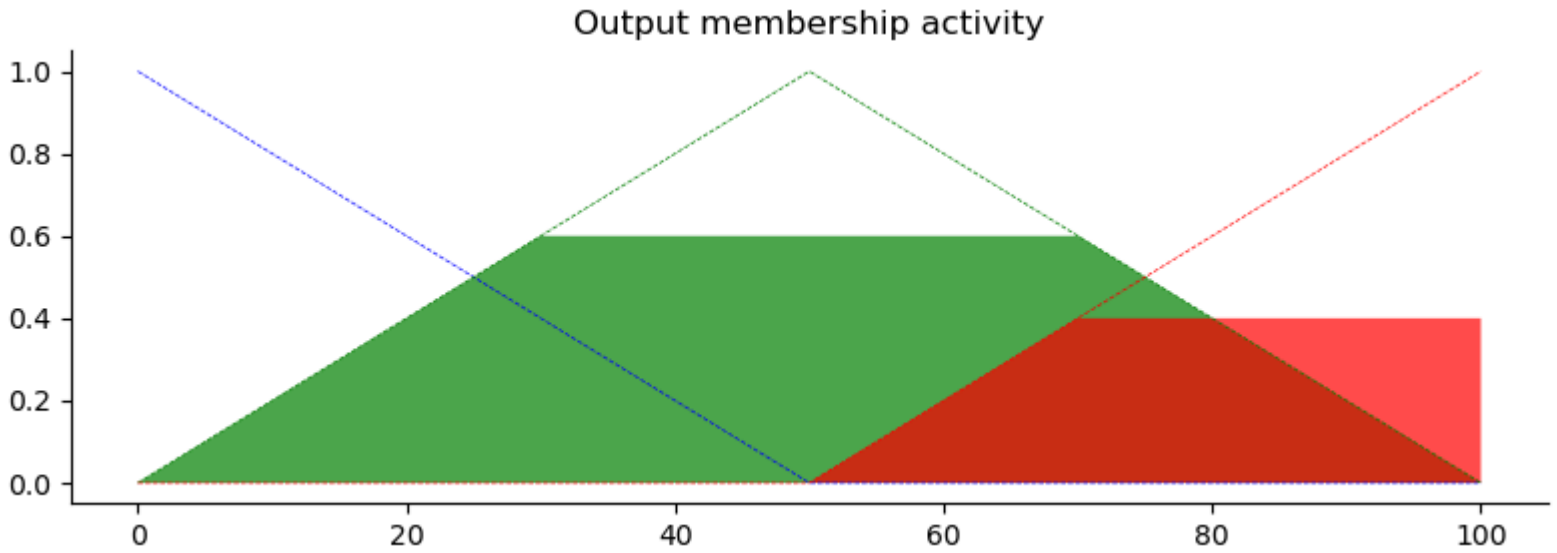
Assume that the humidity is 30% and the temperature is 18°C. We will compute the membership degree for these values and apply the rules.

In [9]:

```
# Given input values
humidity_value = 30.0
temp_value = 18.0

# Calculate the degree of membership for each input value
humidity_level_lo = fuzz.interp_membership(x_humidity, humidity_lo, humidity_value)
humidity_level_md = fuzz.interp_membership(x_humidity, humidity_md, humidity_value)
humidity_level_hi = fuzz.interp_membership(x_humidity, humidity_hi, humidity_value)

temp_level_cold = fuzz.interp_membership(x_temp, temp_cold, temp_value)
```

```
temp_level_comfy = fuzz.interp_membership(x_temp, temp_comfy, temp_value)
temp_level_hot = fuzz.interp_membership(x_temp, temp_hot, temp_value)

# Apply rules:
# Rule 1: If humidity is low OR temperature is cold, heating is high
active_rule1 = np.fmax(humidity_level_lo, temp_level_cold)
heating_activation_hi = np.fmin(active_rule1, heating_hi)

# Rule 2: If humidity is medium AND temperature is comfortable, heating is moderate
active_rule2 = np.fmin(humidity_level_md, temp_level_comfy)
heating_activation_md = np.fmin(active_rule2, heating_md)

# Rule 3: If humidity is high OR temperature is hot, heating is low
active_rule3 = np.fmax(humidity_level_hi, temp_level_hot)
heating_activation_lo = np.fmin(active_rule3, heating_lo)

# Visualize the results of rule activation
fig, ax0 = plt.subplots(figsize=(8, 3))

ax0.fill_between(x_heating, np.zeros_like(x_heating), heating_activation_lo, facecolor='b', alpha=0.7)
ax0.plot(x_heating, heating_lo, 'b', linewidth=0.5, linestyle='--')
ax0.fill_between(x_heating, np.zeros_like(x_heating), heating_activation_md, facecolor='g', alpha=0.7)
ax0.plot(x_heating, heating_md, 'g', linewidth=0.5, linestyle='--')
ax0.fill_between(x_heating, np.zeros_like(x_heating), heating_activation_hi, facecolor='r', alpha=0.7)
ax0.plot(x_heating, heating_hi, 'r', linewidth=0.5, linestyle='--')
ax0.set_title('Output membership activity')

for ax in (ax0,):
```

```
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

plt.tight_layout()
plt.show()
```



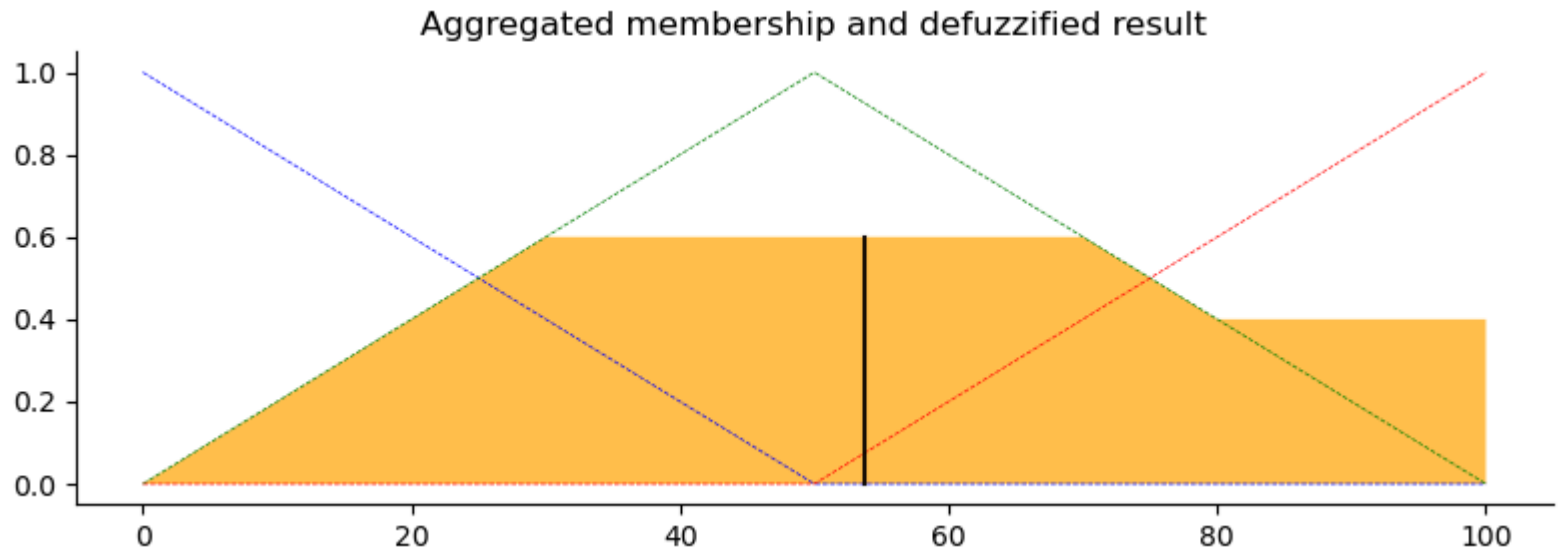

## Rule Aggregation and Defuzzification

To obtain a crisp value for the heating level, we aggregate all the activated output membership functions and perform defuzzification using the centroid method.

In [10]:

```
# Aggregate all rule activations
aggregated = np.fmax(heating_activation_lo,
                     np.fmax(heating_activation_md, heating_activation_hi))
```

```python
# Defuzzify the result
heating = fuzz.defuzz(x_heating, aggregated, 'centroid')
heating_activation = fuzz.interp_membership(x_heating, aggregated, heating)

# Visualize the defuzzified result
fig, ax0 = plt.subplots(figsize=(8, 3))

ax0.plot(x_heating, heating_lo, 'b', linewidth=0.5, linestyle='--')
ax0.plot(x_heating, heating_md, 'g', linewidth=0.5, linestyle='--')
ax0.plot(x_heating, heating_hi, 'r', linewidth=0.5, linestyle='--')
ax0.fill_between(x_heating, np.zeros_like(x_heating), aggregated, facecolor='Orange', alpha=0.7)
ax0.plot([heating, heating], [0, heating_activation], 'k', linewidth=1.5, alpha=0.9)
ax0.set_title('Aggregated membership and defuzzified result')

for ax in (ax0,):
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

plt.tight_layout()
plt.show()

print(f"The defuzzified heating level is: {heating:.2f}%")
```




```
The defuzzified heating level is: 53.77%
```

## Final Thoughts

This example shows the power of fuzzy logic in managing complex decisions based on a few intuitive rules. Even with simple membership functions and a limited number of rules, the system can handle nuanced inputs and generate actionable decisions.