In [12]:
```python
#importing libraries
import numpy as np
```

In [18]:
```python
# The provided Python code defines a class called FuzzySet that deals with fuzzy se
# In a fuzzy set, each element is associated with a membership value that indicates
# with the following key features:

# Constructor: Initializes a fuzzy set with elements and their associated membershi
# with a valid membership probability.

# Fuzzy Set Operations: The class overloads various operators to facilitate standar
    # intersection (&), complement (~), subtraction (-), multiplication (*), and ca

# Helper Function (max_min): The class includes a method called max_min, which comp

# Properties and Methods: The FuzzySet class provides several methods and propertie
# accessing elements using indexing, and iterating over the set's elements.

# Overall, the FuzzySet class allows you to create, manipulate, and perform operati
# empowers you to work effectively with data that possesses uncertainty and impreci
```

In [19]:
```python
class FuzzySet:
    def __init__(self, iterable: any):
        self.f_set = set(iterable)
        self.f_list = list(iterable)
        self.f_len = len(iterable)
        for elem in self.f_set:
            if not isinstance(elem, tuple):
                raise TypeError("No tuples in the fuzzy set")
            if not isinstance(elem[1], float):
                raise ValueError("Probabilities not assigned to elements")

    def __or__(self, other):
        # fuzzy set union
        if len(self.f_set) != len(other.f_set):
            raise ValueError("Length of the sets is different")
        f_set = [x for x in self.f_set]
        other = [x for x in other.f_set]
        return FuzzySet([f_set[i] if f_set[i][1] > other[i][1] else other[i] for i

    def __and__(self, other):
        # fuzzy set intersection
        if len(self.f_set) != len(other.f_set):
            raise ValueError("Length of the sets is different")
        f_set = [x for x in self.f_set]
        other = [x for x in other.f_set]

        return FuzzySet([f_set[i] if f_set[i][1] < other[i][1] else other[i] for i

    def __invert__(self):
        f_set = [x for x in self.f_set]
        for indx, elem in enumerate(f_set):
            f_set[indx] = (elem[0], float(round(1 - elem[1], 2)))
        return FuzzySet(f_set)
    def __sub__(self, other):
        if len(self) != len(other):
            raise ValueError("Length of the sets is different")
        return self & ~other

    def __mul__(self, other):
        if len(self) != len(other):
            raise ValueError("Length of the sets is different")
        return FuzzySet([(self[i][0], self[i][1] * other[i][1]) for i in range(len(

    def __mod__(self, other):
        # cartesian product
        print(f'The size of the relation will be: {len(self)}x{len(other)} ')
        mx = self
        mi = other
        tmp = [[] for i in range(len(mx))]
        i = 0
        for x in mx:
            for y in mi:
                tmp[i].append(min(x[1], y[1]))
            i += 1
        return np.array(tmp)

    @staticmethod
    def max_min(array1: np.ndarray, array2: np.ndarray):
        tmp = np.zeros((array1.shape[0], array2.shape[1]))
        t = list()
        for i in range(len(array1)):
            for j in range(len(array2[0])):
                for k in range(len(array2)):
                    t.append(round(min(array1[i][k], array2[k][j]), 2))
                tmp[i][j] = max(t)
                t.clear()
        return tmp

    def __len__(self):
        self.f_len = sum([1 for i in self.f_set])
        return self.f_len

    def __str__(self):
        return f'{[x for x in self.f_set]}'

    def __getitem__(self, item):
        return self.f_list[item]

    def __iter__(self):
        for i in range(len(self)):
            yield self[i]
```

In [15]:

```
#FuzzySet class and creates fuzzy sets with associated membership values.
#Fuzzy sets are used in fuzzy logic to represent and work with uncertain and imprec
a = FuzzySet({('x1', 0.5), ('x2', 0.7), ('x3', 0.0)})
b = FuzzySet({('x1', 0.8), ('x2', 0.2), ('x3', 1.0)})
c = FuzzySet({('x', 0.3), ('y', 0.3), ('z', 0.5)})
```

```
x = FuzzySet({('a', 0.5), ('b', 0.3), ('c', 0.7)})
y = FuzzySet({('a', 0.6), ('b', 0.4)})
```

In [20]:

```
# Create instances of the FuzzySet class and perform fuzzy set operations.

# Create a fuzzy set 'a' with elements ('x1', 0.5), ('x2', 0.7), and ('x3', 0.0).
a = FuzzySet({('x1', 0.5), ('x2', 0.7), ('x3', 0.0)})

# Create a fuzzy set 'b' with elements ('x1', 0.8), ('x2', 0.2), and ('x3', 1.0).
b = FuzzySet({('x1', 0.8), ('x2', 0.2), ('x3', 1.0)})

# Create a fuzzy set 'c' with elements ('x', 0.3), ('y', 0.3), and ('z', 0.5).
c = FuzzySet({('x', 0.3), ('y', 0.3), ('z', 0.5)})

# Create a fuzzy set 'x' with elements ('a', 0.5), ('b', 0.3), and ('c', 0.7).
x = FuzzySet({('a', 0.5), ('b', 0.3), ('c', 0.7)})

# Create a fuzzy set 'y' with elements ('a', 0.6) and ('b', 0.4).
y = FuzzySet({('a', 0.6), ('b', 0.4)})

# Perform fuzzy set operations and display the results.

# Display the elements and membership values of fuzzy set 'a'.
print(f'a -> {a}')

# Display the elements and membership values of fuzzy set 'b'.
print(f'b -> {b}')

# Display the result of the fuzzy union operation between sets 'a' and 'b'.
print(f'Fuzzy union: \n{a | b}')

# Display the result of the fuzzy intersection operation between sets 'a' and 'b'.
print(f'Fuzzy intersection: \n{a & b}')

# Display the result of inverting the membership values of fuzzy set 'b'.
print(f'Fuzzy inversion of b: \n{~b}')

# Display the result of inverting the membership values of fuzzy set 'a'.
print(f"Fuzzy inversion of a: \n{~a}")

# Display the result of the fuzzy subtraction operation between sets 'a' and 'b'.
print(f'Fuzzy Subtraction: \n{a - b}')
```

```
a -> [('x1', 0.5), ('x3', 0.0), ('x2', 0.7)]
b -> [('x1', 0.8), ('x2', 0.2), ('x3', 1.0)]
Fuzzy union:
[('x1', 0.8), ('x2', 0.2), ('x3', 1.0)]
Fuzzy intersection:
[('x1', 0.5), ('x3', 0.0), ('x2', 0.7)]
Fuzzy inversion of b:
[('x1', 0.2), ('x3', 0.0), ('x2', 0.8)]
Fuzzy inversion of a:
[('x1', 0.5), ('x3', 1.0), ('x2', 0.3)]
Fuzzy Subtraction:
[('x1', 0.2), ('x3', 0.0), ('x2', 0.7)]
```

```
In [17]: r = np.array([[0.6, 0.6, 0.8, 0.9], [0.1, 0.2, 0.9, 0.8], [0.9, 0.3, 0.4, 0.8], [0.
         s = np.array([[0.1, 0.2, 0.7, 0.9], [1.0, 1.0, 0.4, 0.6], [0.0, 0.0, 0.5, 0.9], [0.
         print(f"Max Min of : \n{r} \n\nand \n \n{s}\n\nis:\n")
         print(FuzzySet.max_min(r, s))
```

```
Max Min of :
[[0.6 0.6 0.8 0.9]
 [0.1 0.2 0.9 0.8]
 [0.9 0.3 0.4 0.8]
 [0.9 0.8 0.1 0.2]]

and

[[0.1 0.2 0.7 0.9]
 [1.  1.  0.4 0.6]
 [0.  0.  0.5 0.9]
 [0.9 1.  0.8 0.2]]

is:

[[0.9 0.9 0.8 0.8]
 [0.8 0.8 0.8 0.9]
 [0.8 0.8 0.8 0.9]
 [0.8 0.8 0.7 0.9]]
```

```
In [ ]:
```