In [1]:

```
import pandas as pd
import numpy as np
from math import log2
import pprint
```

In [2]:

```
data = pd.DataFrame([
    ['Sunny', 'Hot', 'High', 'Weak', 'No'],
    ['Sunny', 'Hot', 'High', 'Strong', 'No'],
    ['Overcast', 'Hot', 'High', 'Weak', 'Yes'],
    ['Rain', 'Mild', 'High', 'Weak', 'Yes'],
    ['Rain', 'Cool', 'Normal', 'Strong', 'Yes'],
    ['Rain', 'Cool', 'Normal', 'Weak', 'Yes'],
    ['Overcast', 'Cool', 'Normal', 'Strong', 'Yes'],
    ['Sunny', 'Mild', 'High', 'Weak', 'No'],
    ['Sunny', 'Cool', 'Normal', 'Weak', 'Yes'],
    ['Rain', 'Mild', 'Normal', 'Weak', 'Yes'],
    ['Sunny', 'Mild', 'Normal', 'Strong', 'Yes'],
    ['Overcast', 'Mild', 'High', 'Strong', 'Yes'],
    ['Overcast', 'Hot', 'Normal', 'Weak', 'Yes'],
    ['Rain', 'Mild', 'High', 'Strong', 'No']
], columns=['Outlook', 'Temperature', 'Humidity', 'Wind', 'PlayTennis'])
```

In [3]:

```
def entropy(target_col):
    values, counts = np.unique(target_col, return_counts=True)
    entropy = sum([-p * log2(p) for p in counts / counts.sum()])
    return entropy
```

In [4]:

```
def info_gain(data, split_attribute_name, target_name="PlayTennis"):
    total_entropy = entropy(data[target_name])
    vals, counts = np.unique(data[split_attribute_name], return_counts=True)

    weighted_entropy = sum(
        (counts[i] / sum(counts)) *
        entropy(data[data[split_attribute_name] == vals[i]][target_name])
        for i in range(len(vals))
    )

    information_gain = total_entropy - weighted_entropy
    return information_gain
```

In [5]:

```
def ID3(data, original_data, features, target_attribute_name="PlayTennis", parent_node_class=None):
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data) == 0:
        return np.unique(original_data[target_attribute_name])[np.argmax(
            np.unique(original_data[target_attribute_name], return_counts=True)[1])]
    elif len(features) == 0:
        return parent_node_class
    else:
        parent_node_class = np.unique(data[target_attribute_name])[np.argmax(
            np.unique(data[target_attribute_name], return_counts=True)[1])]

        item_values = [info_gain(data, feature, target_attribute_name) for feature in features]
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]

        tree = {best_feature: {}}
        for value in np.unique(data[best_feature]):
            sub_data = data[data[best_feature] == value]
            new_features = features[:best_feature_index] + features[best_feature_index + 1:]

            subtree = ID3(sub_data, original_data, new_features, target_attribute_name, parent_node_class)
            tree[best_feature][value] = subtree

        return tree
```

In [6]:

```
features = list(data.columns)
features.remove("PlayTennis")
tree = ID3(data, data, features)
```

In [7]:

```
# Print tree
import pprint
pprint.pprint(tree)
```

```
{'Humidity': {'High': {'Outlook': {'Overcast': 'Yes',
                                   'Rain': {'Wind': {'Strong': 'No',
                                                     'Weak': 'Yes'}},
                                   'Sunny': 'No'}},
              'Normal': 'Yes'}}
```

In [8]:

```
def predict(query, tree, default='Yes'):
    for attr in query:
        if attr in tree:
            try:
                result = tree[attr][query[attr]]
            except:
                return default
            if isinstance(result, dict):
                return predict(query, result)
            else:
                return result
    return default
```

In [9]:

```
sample = {'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'High', 'Wind': 'Strong'}
prediction = predict(sample, tree)
print("\nPredicted Output for sample is:", prediction)
```

```
Predicted Output for sample is: No
```

In [ ]: