1. pulp Installation:

In [1]:
```
!pip install pulp
from pulp import *
import numpy as np
import pandas as pd
```

```
Collecting pulp
  Downloading PuLP-2.9.0-py3-none-any.whl.metadata (5.4 kB)
Downloading PuLP-2.9.0-py3-none-any.whl (17.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.7/17.7 MB 41.1 MB/s eta 0:00:00
Installing collected packages: pulp
Successfully installed pulp-2.9.0
```

PuLP also comes with the CBC solver, which is an open-source solver and the default for solving linear problems.

2. Problem Definition:

The first step in any optimization problem is to define the problem. You can either define a minimization or maximization problem using PuLP.

In [2]:
```
#Define a minimization problem
MinModel = LpProblem("MinimizeCost", LpMinimize)

#Define a maximization problem
MaxModel = LpProblem("MaximizeProfit", LpMaximize)
```

The string passed as the first argument ("MinimizeCost" or "MaximizeProfit") is just a name for the problem and is optional.

- LpMaximize: Indicates that the problem aims to maximize the objective function.
- LpMinimize: Indicates that the problem aims to minimize the objective function.

3. Defining Variables:

The next step is to define the decision variables that PuLP will optimize. Decision variables represent quantities you want to determine in the solution.

PuLP uses the LpVariable class to define variables. You can specify:

- Name: The variable name.
- lowBound: The minimum value for the variable (default is $-\infty$) (optional).

- upBound: The maximum value for the variable (default is ∞) (optional).
- cat: The category of the variable, e.g., 'Continuous' (default) or 'Integer'.

```
In [ ]: #Defining a continuous variable x
        x = LpVariable('x', lowBound=0, cat='Continuous')

        #Defining an integer variable y between 0 and 10
        y = LpVariable('y', lowBound=0, upBound=10, cat='Integer')

        #Defining a binary variable z (0 or 1)
        z = LpVariable('z', cat='Binary')
```

Defining Multiple Variables:

When there are multiple variables (such as indexed variables), you can use `LpVariable.dicts` :

```
In [3]: #Define multiple decision variables x_ij using LpVariable.dicts
        indices = ['A', 'B', 'C']

        d = LpVariable.dicts('d', indices, lowBound=0, cat='Continuous')

        d
```

```
Out[3]: {'A': c_A, 'B': c_B, 'C': c_C}
```

```
In [4]: crop = ['sugar beets', 'cotton', 'sorghum']
        area = [1, 2, 3]

        x = LpVariable.dicts('x', ((c, k) for c in crop for k in area), lowBound=0, cat='Continuous')

        x
```

```
Out[4]: {('sugar beets', 1): x_('sugar_beets',_1),
         ('sugar beets', 2): x_('sugar_beets',_2),
         ('sugar beets', 3): x_('sugar_beets',_3),
         ('cotton', 1): x_('cotton',_1),
         ('cotton', 2): x_('cotton',_2),
         ('cotton', 3): x_('cotton',_3),
         ('sorghum', 1): x_('sorghum',_1),
         ('sorghum', 2): x_('sorghum',_2),
         ('sorghum', 3): x_('sorghum',_3)}
```

4. Objective Function

The objective function is the function you want to optimize (either maximize or minimize). The objective is defined using `+=` to add terms to the problem object.

```
In [ ]: #Define a maximization objective function
        MaxModel += 2*x + 3*y
```

In the above code, $2x + 3y$ is the linear expression that represents the objective function.

5. Constraints:

After defining the objective function, you add constraints to the problem. Constraints define the restrictions on the variables (e.g., limits on resources, capacities, or other conditions).

In [ ]:
```
#Adding constraints to the problem
MaxModel += x + y <= 10, "Constraint 1"
MaxModel += 2*x - y >= 5
```

$x + y <= 10$ is a constraint that limits the values of x and y. The string ("Constraint 1") is the name of the constraint, which is optional but useful for debugging.

Constraints with Indexed Variables:

For problems with multiple variables (like indexed variables), you can use loops to add constraints:

In [ ]:
```
#Define constraints for each index
for i in indices:
    MaxModel += x[i] >= 10
```

6. Solving the Problem:

After defining the objective function and constraints, you can solve the problem using the `solve()` method. PuLP uses CBC solver by default, but you can specify another solver if installed.

In [ ]:
```
MaxModel.solve()
```

You can check the status of the solution with LpStatus:

In [ ]:
```
print(f"Status: {LpStatus[MaxModel.status]}")
```

Common statuses:

- Optimal: The problem has been solved optimally.
- Infeasible: There are no feasible solutions.
- Unbounded: The solution is unbounded (i.e., the objective can grow infinitely).

7. Extracting Results

Once the problem is solved, you can extract the optimal values of the decision variables using their `.varValue` attribute.

In [ ]:
```
#Print variable values
for v in MaxModel.variables():
    print(f"{v.name} = {v.varValue}")

#Print objective function value
print(f"Objective value: {pulp.value(MaxModel.objective)}")
```

8. pulp functions

Summary of PuLP Functions:

Function	Description
lpSum	Sums a list of linear expressions or variables.
lpDot	Computes the dot product of two lists (e.g., coefficients and variables).
lpMax	Computes the maximum of a list of expressions or variables.
lpMin	Computes the minimum of a list of expressions or variables.
value	Returns the value of a variable, expression, or objective after solving the problem.
makeDict	Creates a dictionary with default values, often for multi-dimensional index variables.
lpAffineExpression	Internal representation of linear expressions. Users do not call this directly.

- **lpSum()**: The `lpSum` function is used to compute the sum of a list of linear expressions or variables. It is often used to simplify the creation of objective functions and constraints by allowing you to sum a collection of terms.

In [5]:
```
#Define the problem
problem = LpProblem("ExampleProblem", LpMaximize)

#Define decision variables
x = LpVariable.dicts('x', [1, 2, 3], lowBound=0)

#Objective function: Maximize the sum of all x variables
problem += lpSum(x[i] for i in [1, 2, 3])
```

In [7]:
```
problem.objective
```

Out[7]:
```
1*x_1 + 1*x_2 + 1*x_3 + 0
```

- **lpDot()**: The `lpDot` function is used to compute the dot product of two lists. This is particularly useful when you need to multiply a list of coefficients by a list of decision variables.

In [8]:
```
#Define the problem
problem = LpProblem("CostMinimization", LpMinimize)

#Define decision variables
x = LpVariable.dicts('x', [1, 2, 3], lowBound=0)

#Cost coefficients for each item
costs = [10, 20, 15]

#Objective function: Minimize total cost (dot product of costs and quantities)
problem += lpDot(costs, [x[i] for i in [1, 2, 3]])
```

In [9]:
```
problem.objective
```

Out[9]:
```
10*x_1 + 20*x_2 + 15*x_3 + 0
```