## 0. Imports and Setting up Anthropic API Client

```
from google.colab import drive

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install python-dotenv

import os
import dotenv

dotenv.load_dotenv('/content/drive/MyDrive/.env')
```

```
Collecting python-dotenv
  Downloading python_dotenv-1.0.1-py3-none-any.whl (19 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.0.1
True
```

```
# Load Prompts and Problem Description
prompt1_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt1_MathematicalModel.txt'
prompt2_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt2_PyomoCode.txt'
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/NL/NL2.txt'

prompt1_file = open(prompt1_path, "r")
prompt2_file = open(prompt2_path, "r")
problem_desc_file = open(problem_desc_path, "r")

prompt1 = prompt1_file.read()
print("Prompt 1:\n", prompt1)

prompt2 = prompt2_file.read()
print("Prompt 2:\n", prompt2)

problem_desc = problem_desc_file.read()
print("Problem Description:\n", problem_desc)
```

```
Prompt 1:
 Please write a mathematical optimization model for this problem. Include parameters, decision variables, the objective
Prompt 2:
 Please write a python pyomo code for this optimization problem.
Use sample data where needed.
Indicate where you use sample data.
Problem Description:
 You are a chemist in charge of a process that requires two liquids to reach the end product. Your goal is to maximize t
```

```
!pip install anthropic
```

Show hidden output

```
# Importing Anthropic & Setting Headers
import anthropic

client = anthropic.Anthropic(
    # defaults to os.environ.get("ANTHROPIC_API_KEY")
    api_key=os.environ.get("ANTHROPIC_API_KEY"),
)
```

## 1. Prompt 1: Create Mathematical Model

```
message = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt1,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)

response_p1 = message.content[0].text
```

```
# Print response
print(response_p1)
```

```
Let's formulate the mathematical optimization model for this problem by defining the parameters, decision variables, obj

Parameters:
- L: Maximum capacity of the mixture tank (80 l)

Decision Variables:
- x₁: Litres of the first component used
- x₂: Litres of the second component used

Objective Function:
Maximize the quality of the product, which is measured by the difference between the Napierian logarithm of the product

Maximize: ln(x₁ × x₂) - ∛(x₁ + 2x₂)

Constraints:
1. Stability constraint: The squared difference between the litres employed of each component and the mean litres utilis
   (x₁ - (x₁ + x₂) / 2)² + (x₂ - (x₁ + x₂) / 2)² ≤ 10

2. Mixture tank capacity constraint: The sum of the litres of both components should not exceed the maximum capacity of
   x₁ + x₂ ≤ L

3. Non-negativity constraints: The litres of each component used should be non-negative.
   x₁ ≥ 0
   x₂ ≥ 0

Complete Mathematical Optimization Model:
Maximize: ln(x₁ × x₂) - ∛(x₁ + 2x₂)

Subject to:
1. (x₁ - (x₁ + x₂) / 2)² + (x₂ - (x₁ + x₂) / 2)² ≤ 10
2. x₁ + x₂ ≤ 80
3. x₁ ≥ 0
4. x₂ ≥ 0

Where:
- x₁: Litres of the first component used
- x₂: Litres of the second component used

This mathematical optimization model aims to maximize the quality of the product while satisfying the stability constrai
```

## 2. Prompt 2: Write Code based on Mathematical Model

```
message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": response_p1}
    ]
)

response_p2 = message2.content[0].text
```

```
# Print response
print(response_p2)
```

Here's the Python code using the Pyomo library to solve the optimization problem:

```python
from pyomo.environ import *
import math

# Create a Pyomo model
model = ConcreteModel()

# Define decision variables
model.x1 = Var(domain=NonNegativeReals)
model.x2 = Var(domain=NonNegativeReals)

# Define objective function
model.obj = Objective(expr=log(model.x1 * model.x2) - (model.x1 + 2 * model.x2)**(1/3), sense=maximize)

# Define constraints
model.stability_constraint = Constraint(expr=(model.x1 - (model.x1 + model.x2) / 2)**2 + (model.x2 - (model.x1 + model.x
model.capacity_constraint = Constraint(expr=model.x1 + model.x2 <= 80)

# Solve the optimization problem
solver = SolverFactory('ipopt')
solver.solve(model)
```

```
# Print the results
print("Optimal solution:")
print("x1 =", value(model.x1))
print("x2 =", value(model.x2))
print("Objective value =", value(model.obj))
```

```
Explanation:
1. We import the necessary modules from the Pyomo library and the `math` module.

2. We create a concrete Pyomo model using `ConcreteModel()`.

3. We define the decision variables `x1` and `x2` as non-negative real numbers using `Var(domain=NonNegativeReals)`.

4. We define the objective function using `Objective()`. The objective function is to maximize the quality of the produc

5. We define the constraints using `Constraint()`:
   - The stability constraint is expressed as `(model.x1 - (model.x1 + model.x2) / 2)**2 + (model.x2 - (model.x1 + model
   - The mixture tank capacity constraint is expressed as `model.x1 + model.x2 <= 80`.

6. We create an instance of the IPOPT solver using `SolverFactory('ipopt')` and solve the optimization problem using `so

7. Finally, we print the optimal solution values for `x1`, `x2`, and the objective value.

Note that no sample data is used in this code as all the necessary parameters are directly defined within the code itsel
```

## 4. Input Problem Data and Test Model Code

+ Code + Text

```
# Installing pyomo nd solver
!pip install -q pyomo
!wget -N -q "https://matematica.unipv.it/gualandi/solvers/ipopt-linux64.zip"
!unzip -o -q ipopt-linux64
```

Show hidden output

```
from pyomo.environ import *
import math

# Create a Pyomo model
model = ConcreteModel()

# Define decision variables
model.x1 = Var(domain=NonNegativeReals)
model.x2 = Var(domain=NonNegativeReals)

# Define objective function
model.obj = Objective(expr=log(model.x1 * model.x2) - (model.x1 + 2 * model.x2)**(1/3), sense=maximize)

# Define constraints
model.stability_constraint = Constraint(expr=(model.x1 - (model.x1 + model.x2) / 2)**2 + (model.x2 - (model.x1 + model.x2) /
model.capacity_constraint = Constraint(expr=model.x1 + model.x2 <= 80)

# Solve the optimization problem
solver = SolverFactory('ipopt')
solver.solve(model)

# Print the results
print("Optimal solution:")
print("x1 =", value(model.x1))
print("x2 =", value(model.x2))
print("Objective value =", value(model.obj))
```

```
Optimal solution:
x1 = 42.236067853543105
x2 = 37.76393234768411
Objective value = 2.473033919646447
```

## 5. Correct The Model Code to Test Mathematical Model (if applicable)