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 the given problem. We'll define the parameters, decision variabl

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

Decision Variables:
- x₁: Litres of the first liquid component used
- x₂: Litres of the second liquid 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. Capacity constraint: The total litres of both components should not exceed the mixture tank's capacity (80 l).
   x₁ + x₂ ≤ 80

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

The complete mathematical optimization model:

Maximize:
ln(x₁ × x₂) - ∛(x₁ + 2x₂)

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

This model aims to maximize the quality of the product while satisfying the stability and capacity constraints, ensuring
```

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 a Python code using Pyomo to solve the optimization problem:

```python
from pyomo.environ import *
from math import log, sqrt

# Create a Pyomo model
model = ConcreteModel()

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

# Define the objective function
def objective_rule(model):
    return log(model.x1 * model.x2) - (model.x1 + 2 * model.x2)**(1/3)

model.objective = Objective(rule=objective_rule, sense=maximize)

# Define constraints
def stability_constraint_rule(model):
    mean_litres = (model.x1 + model.x2) / 2
    return (model.x1 - mean_litres)**2 + (model.x2 - mean_litres)**2 <= 10

model.stability_constraint = Constraint(rule=stability_constraint_rule)
````
````

```
def capacity_constraint_rule(model):
    return model.x1 + model.x2 <= 80

model.capacity_constraint = Constraint(rule=capacity_constraint_rule)

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

# Print the results
print("Optimal solution:")
print(f"x1 = {value(model.x1):.2f} litres")
print(f"x2 = {value(model.x2):.2f} litres")
print(f"Objective value: {value(model.objective):.2f}")
```

```

In this code, we:

1. Import the necessary Pyomo components and mathematical functions.
2. Create a Pyomo `ConcreteModel`.
3. Define the decision variables `x1` and `x2` as non-negative real numbers.
4. Define the objective function using the `objective_rule` and add it to the model with `sense=maximize`.
5. Define the stability constraint using the `stability_constraint_rule` and add it to the model.
6. Define the capacity constraint using the `capacity_constraint_rule` and add it to the model.
7. Solve the optimization problem using the IPOPT solver (you may need to install it separately).
8. Print the optimal solution values for `x1`, `x2`, and the objective value.

Note: In this code, we don't use any sample data as the problem is self-contained and doesn't require external data. The

To run this code, make sure you have Pyomo and the IPOPT solver installed in your Python environment. You can install Py
```

4. Input Problem Data and Test Model Code

```
# 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
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 31.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.6/49.6 kB 5.3 MB/s eta 0:00:00
```

```
from pyomo.environ import *
import math # MODIFIED IMPORT SO CODE EXECUTES

# Create a Pyomo model
model = ConcreteModel()

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

# Define the objective function
def objective_rule(model):
    return log(model.x1 * model.x2) - (model.x1 + 2 * model.x2)**(1/3)

model.objective = Objective(rule=objective_rule, sense=maximize)

# Define constraints
def stability_constraint_rule(model):
    mean_litres = (model.x1 + model.x2) / 2
    return (model.x1 - mean_litres)**2 + (model.x2 - mean_litres)**2 <= 10

model.stability_constraint = Constraint(rule=stability_constraint_rule)

def capacity_constraint_rule(model):
    return model.x1 + model.x2 <= 80

model.capacity_constraint = Constraint(rule=capacity_constraint_rule)

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

# Print the results
print("Optimal solution:")
print(f"x1 = {value(model.x1):.2f} litres")
print(f"x2 = {value(model.x2):.2f} litres")
print(f"Objective value: {value(model.objective):.2f}")
```

```
Optimal solution:
x1 = 42.24 litres
x2 = 37.76 litres
Objective value: 2.47
```

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