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 will define the parameters, decision varia

Parameters:
- Let the maximum capacity of the mixture tank be denoted as C, which is 80 liters.

Decision Variables:
- Let x₁ be the number of liters of the first liquid component used.
- Let x₂ be the number of liters of the second liquid component used.

Objective Function:
- The quality of the product is measured by the difference between the Napierian logarithm of the product of the liters
- Maximize: ln(x₁ × x₂) - ∛(x₁ + 2x₂)

Constraints:
1. Stability Constraint:
   - The squared difference between the liters used of each component and the mean liters used of both components should
   - (x₁ - (x₁ + x₂) / 2)² + (x₂ - (x₁ + x₂) / 2)² ≤ 10

2. Capacity Constraint:
   - The sum of the liters used of both components should not exceed the maximum capacity of the mixture tank.
   - x₁ + x₂ ≤ C

3. Non-negativity Constraints:
   - The number of liters used for each component should be non-negative.
   - x₁ ≥ 0
   - x₂ ≥ 0

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

This optimization model aims to maximize the quality of the product while satisfying the stability constraint, capacity
```

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)
```

```
# Create the model
model = ConcreteModel()

# Define the parameters
model.C = Param(initialize=80)  # Maximum capacity of the mixture tank

# Define the decision variables
model.x1 = Var(domain=NonNegativeReals)  # Liters of the first liquid component
model.x2 = Var(domain=NonNegativeReals)  # Liters of the second liquid component

# Define the objective function
def quality_rule(model):
    return log(model.x1 * model.x2) - (model.x1 + 2 * model.x2) ** (1/3)
model.quality = Objective(rule=quality_rule, sense=maximize)

# Define the constraints
def stability_rule(model):
    mean = (model.x1 + model.x2) / 2
    return (model.x1 - mean)**2 + (model.x2 - mean)**2 <= 10
model.stability = Constraint(rule=stability_rule)
```

```
    solver.solve(model)

    # Print the results
    print("Optimal Solution:")
    print("x1 =", value(model.x1))
    print("x2 =", value(model.x2))
    print("Quality =", value(model.quality))
    ```

    Explanation:

    1. We import the necessary components from the `pyomo.environ` module.

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

    3. We define the parameter `C` to represent the maximum capacity of the mixture tank, which is set to 80 liters.

    4. We define the decision variables `x1` and `x2` to represent the liters of the first and second liquid components, r

    5. We define the objective function `quality_rule` using the mathematical expression provided. The objective is to max

    6. We define the stability constraint `stability_rule` using the given inequality. It ensures that the squared differe

    7. We define the capacity constraint `capacity_rule` to ensure that the sum of the liters used of both components does

    8. We create an instance of the solver using `SolverFactory('ipopt')`. You can choose a different solver if desired.

    9. We solve the optimization problem using `solver.solve(model)`.

    10. Finally, we print the optimal solution, including the values of `x1`, `x2`, and the quality objective.
```

## 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 52.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.6/49.6 kB 5.0 MB/s eta 0:00:00
```

```
from pyomo.environ import *

# Create the model
model = ConcreteModel()

# Define the parameters
model.C = Param(initialize=80)  # Maximum capacity of the mixture tank

# Define the decision variables
model.x1 = Var(domain=NonNegativeReals)  # Liters of the first liquid component
model.x2 = Var(domain=NonNegativeReals)  # Liters of the second liquid component

# Define the objective function
def quality_rule(model):
    return log(model.x1 * model.x2) - (model.x1 + 2 * model.x2) ** (1/3)
model.quality = Objective(rule=quality_rule, sense=maximize)

# Define the constraints
def stability_rule(model):
    mean = (model.x1 + model.x2) / 2
    return (model.x1 - mean)**2 + (model.x2 - mean)**2 <= 10
model.stability = Constraint(rule=stability_rule)

def capacity_rule(model):
    return model.x1 + model.x2 <= model.C
model.capacity = Constraint(rule=capacity_rule)

# 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("Quality =", value(model.quality))
```

```
Optimal Solution:
x1 = 42.236067853543105
x2 = 37.76393234768411
```
```

```
Quality = 2.473033919646447
```

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