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
# Variables Prompt
prompt11_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt11_MathematicalModel.txt'

# Objective Prompt
prompt12_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt12_MathematicalModel.txt'

# Constraint Prompt
prompt13_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt13_MathematicalModel.txt'

# Code Prompt
prompt2_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt2_PyomoCode.txt'
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/NL/NL2.txt'

prompt11_file = open(prompt11_path, "r")
prompt12_file = open(prompt12_path, "r")
prompt13_file = open(prompt13_path, "r")
prompt2_file = open(prompt2_path, "r")
problem_desc_file = open(problem_desc_path, "r")

prompt11 = prompt11_file.read()
print("Prompt 1.1 (Variables):\n", prompt11)

prompt12 = prompt12_file.read()
print("Prompt 1.2 (Objctive):\n", prompt12)

prompt13 = prompt13_file.read()
print("Prompt 1.3 (Constraints):\n", prompt13)

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

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

```
Prompt 1.1 (Variables):
 Please formulate only the variables for this mathematical optimization problem.
Prompt 1.2 (Objctive):
 Please formulate only the objective function for this mathematical optimization problem.
Prompt 1.3 (Constraints):
 Please formulate only the constraints for this mathematical optimization problem.
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
```

```
Collecting anthropic
  Downloading anthropic-0.28.0-py3-none-any.whl (862 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 862.7/862.7 kB 11.3 MB/s eta 0:00:00
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from anthropic) (1.7.0)
Collecting httpx<1,>=0.23.0 (from anthropic)
```

```
  Downloading httpx-0.27.0-py3-none-any.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 10.7 MB/s eta 0:00:00
Collecting jiter<1,>=0.4.0 (from anthropic)
  Downloading jiter-0.4.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (328 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 328.3/328.3 kB 27.8 MB/s eta 0:00:00
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (2.7.3)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from anthropic) (1.3.1)
Requirement already satisfied: tokenizers>=0.13.0 in /usr/local/lib/python3.10/dist-packages (from anthropic) (0.19.1)
Requirement already satisfied: typing-extensions<5,>=4.7 in /usr/local/lib/python3.10/dist-packages (from anthropic) (4.
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->anthropic) (3
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->anthropi
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->anthropic) (20
Collecting httpcore==1.* (from httpx<1,>=0.23.0->anthropic)
  Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 10.2 MB/s eta 0:00:00
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->anthropic)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 7.2 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.
Requirement already satisfied: pydantic-core==2.18.4 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/lib/python3.10/dist-packages (from tokenizers>
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->t
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->t
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->huggi
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface
Installing collected packages: jiter, h11, httpcore, httpx, anthropic
Successfully installed anthropic-0.28.0 h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 jiter-0.4.1
```

```
# 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 1.1: Create Variables for Mathematical Model

```
message11 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt11,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)

response_p11 = message11.content[0].text


# Print response
print(response_p11)
```

```
To formulate the mathematical optimization model, let's define the decision variables and the given information:

Decision variables:
Let x₁ be the number of litres of the first liquid component.
Let x₂ be the number of litres of the second liquid component.

Given information:
- The quality of the product is measured by ln(x₁ * x₂) - ∛(x₁ + 2x₂).
- The squared difference between the litres of each component and the mean litres used of both components should be less
- The mixture tank has a capacity of 80 litres.

Now, let's formulate the optimization problem:

Objective function:
Maximize the quality of the product: ln(x₁ * x₂) - ∛(x₁ + 2x₂)

Constraints:
1. Stability constraint: (x₁ - (x₁ + x₂)/2)² + (x₂ - (x₁ + x₂)/2)² ≤ 10
2. Capacity constraint: x₁ + x₂ ≤ 80
3. Non-negativity constraints: x₁ ≥ 0, x₂ ≥ 0

The complete mathematical optimization model can be written as:

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 find the optimal values of x₁ and x₂ that maximize the quality of the product while satisfying the st
```

1. Prompt 1.2: Create Objective for Mathematical Model

```
message12 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt12,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11)}
    ]
)

response_p12 = message12.content[0].text


# Print response
print(response_p12)
```

```
Maximize ln(x₁ * x₂) - ∛(x₁ + 2x₂)
```

1. Prompt 1.3: Create Constraints for Mathematical Model

```
message13 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt13,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": (problem_desc + response_p11 + response_p12)}
    ]
)

response_p13 = message13.content[0].text


# Print response
print(response_p13)
```

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

2. Prompt 2: Write Code based on Mathematical Model

In case more than the component of each step was generated, use "last instance"/ "newest version" of the components.

Example: Prompt 1.1 (Variables): Model generates only variables Prompt 1.2 (Objectvie): Model generates objective and constraints Prompt 1.3 (Constraints): Model generates only constraints

Use Variables from step 1.1, use objective from 1.2 and use constraints from 1.3 (since constraints version 1.3 is more recent than constraints version 1.2).

This set up allows for potential self-correction of the model throughout the three prompts.

Correctness of each component is still generated in the respective step, so self-correction is independent of this.

```
response_p11_adjusted = """
To formulate the mathematical optimization model, let's define the decision variables and the given information:

Decision variables:
Let x₁ be the number of litres of the first liquid component.
Let x₂ be the number of litres of the second liquid component.

Given information:
- The quality of the product is measured by ln(x₁ * x₂) - ∛(x₁ + 2x₂).
- The squared difference between the litres of each component and the mean litres used of both components should be less tha
- The mixture tank has a capacity of 80 litres.

Now, let's formulate the optimization problem:
"""


message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": (response_p11_adjusted + response_p12 + response_p13)}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

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

    ```python
    from pyomo.environ import *

    # Create a Pyomo model
    model = ConcreteModel()

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

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

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

    # Solve the optimization problem
    solver = SolverFactory('ipopt')  # You can use other solvers like 'glpk' or 'cplex' if available
    solver.solve(model)

    # Print the optimal solution
    print("Optimal solution:")
    print("x1 =", model.x1.value)
    print("x2 =", model.x2.value)
    print("Quality =", model.quality.expr())
    ```

    In this code:
    1. We import the necessary components from the Pyomo library.
    2. We create a Pyomo model using `ConcreteModel()`.
    3. We define the decision variables `x1` and `x2` as non-negative real variables using `Var(within=NonNegativeReals)`.
    4. We define the objective function `quality` to maximize the expression `ln(x₁ * x₂) - ∛(x₁ + 2x₂)` using `Objective(ex
    5. We define the stability constraint using `Constraint(expr=...)` to ensure that the squared difference between the lit
    6. We define the capacity constraint to ensure that the sum of `x1` and `x2` is less than or equal to 80.
    7. We create a solver instance using `SolverFactory()`. In this example, we use the 'ipopt' solver, but you can choose o
    8. We solve the optimization problem using `solver.solve(model)`.
    9. Finally, we print the optimal values of `x1`, `x2`, and the corresponding quality value.

    Note that this code assumes you have the Pyomo library and a suitable solver (e.g., IPOPT) installed. You may need to in

    Also, please note that this code does not use any sample data. The optimization problem is solved based on the given mat
```

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 76.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.6/49.6 kB 5.7 MB/s eta 0:00:00
```

```
from pyomo.environ import *

# Create a Pyomo model
model = ConcreteModel()

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

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

# Define constraints
model.stability = Constraint(expr=(model.x1 - (model.x1 + model.x2) / 2)**2 + (model.x2 - (model.x1 + model.x2) / 2)**2 <= 10
model.capacity = Constraint(expr=model.x1 + model.x2 <= 80)

# Solve the optimization problem
solver = SolverFactory('ipopt')  # You can use other solvers like 'glpk' or 'cplex' if available
solver.solve(model)

# Print the optimal solution
print("Optimal solution:")
print("x1 =", model.x1.value)
print("x2 =", model.x2.value)
print("Quality =", model.quality.expr())
```

```
Optimal solution:
x1 = 42.236067853543105
x2 = 37.76393234768411
Quality = 2.473033919646447
```

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