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)
```

```
second component litres. It is required that the squared difference between the litres employed of each component and the
```

```
!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 1.1: 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 objective function:

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.

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

max z = ln(x₁ × x₂) − ∛(x₁ + 2x₂)

Constraints:
1. The squared difference between the litres employed of each component and the mean litres utilised of both components

(x₁ − (x₁ + x₂)/2)² + (x₂ − (x₁ + x₂)/2)² ≤ 10

2. The mixture tank available can only fit 80 l.

x₁ + x₂ ≤ 80

3. Non-negativity constraints:

x₁ ≥ 0
x₂ ≥ 0

In summary, the mathematical optimization model is:

max z = 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
```

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)
```

```
max z = 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 objective function:

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.
"""


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 a Python Pyomo code for the optimization problem:

```python
from pyomo.environ import *

model = ConcreteModel()

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

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

# Constraints
model.constraint1 = Constraint(expr=(model.x1 - (model.x1 + model.x2)/2)**2 + (model.x2 - (model.x1 + model.x2)/2)**2 <=
model.constraint2 = Constraint(expr=model.x1 + model.x2 <= 80)
```
```

```
# Solve the model
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 components from the `pyomo.environ` module.

2. We create a `ConcreteModel` named `model`.

3. We define the decision variables `x1` and `x2` using `Var()` and specify that they should be non-negative using `with

4. We define the objective function using `Objective()` and provide the mathematical expression. We set `sense=maximize`

5. We define the constraints using `Constraint()` and provide the mathematical expressions for each constraint.

6. We create an instance of the solver using `SolverFactory('ipopt')`. In this case, we are using the 'ipopt' solver, wh

7. We solve the model using `solver.solve(model)`.

8. Finally, we print the optimal values of the decision variables `x1` and `x2` using `value()` and the optimal objectiv

Note: This code assumes that you have the Pyomo package and the 'ipopt' solver installed. You may need to install them s

No sample data is used in this code, as the optimization problem is defined using the given mathematical expressions and

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

```
from pyomo.environ import *

model = ConcreteModel()

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

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

# Constraints
model.constraint1 = Constraint(expr=(model.x1 - (model.x1 + model.x2)/2)**2 + (model.x2 - (model.x1 + model.x2)/2)**2 <= 10)
model.constraint2 = Constraint(expr=model.x1 + model.x2 <= 80)

# Solve the model
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)