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/NL1.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:
 A firm that packs refreshments and beers, situated in the province of Valencia (Spain) employs the same syrup to produc
49000 times X1 minus X1 squared plus 30 times X2 minus two times X2 squared.
It costs 150 dollars to buy and process each hectolitre of syrup. The S1 packaging line has a net capacity of producing
```

```
!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)
```

```
The decision variables for this optimization problem are:

X1 = number of units of the 1.5 l COLI product produced daily
X2 = number of units of the 1.5 l PEPSA product produced daily
```

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)
```

```
The objective function for this mathematical optimization problem, which aims to maximize the firm's daily profits, can

Maximize:
(49000 * X1 - X1^2) + (30 * X2 - 2 * X2^2) - 150 * (X1 / 40 + X2 / 20)

Where:
- (49000 * X1 - X1^2) represents the income from producing X1 units of the 1.5 l COLI product
- (30 * X2 - 2 * X2^2) represents the income from producing X2 units of the 1.5 l PEPSA product
- 150 * (X1 / 40 + X2 / 20) represents the cost of buying and processing the syrup needed to produce X1 units of COLI an

The objective function can be simplified to:

Maximize:
49000 * X1 - X1^2 + 30 * X2 - 2 * X2^2 - (150/40) * X1 - (150/20) * X2

Or further simplified to:

Maximize:
49000 * X1 - X1^2 + 30 * X2 - 2 * X2^2 - 3.75 * X1 - 7.5 * X2

This objective function represents the daily profits the firm aims to maximize by determining the optimal number of unit
```

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)
```

```
Based on the given information, the constraints for this mathematical optimization problem can be formulated as follows:

1. Syrup constraint:
   X1 / 40 + X2 / 20 ≤ Syrup_Capacity

   Where Syrup_Capacity is the maximum amount of syrup available per day (in hectolitres). This constraint ensures that

2. Production capacity constraint:
   X1 + X2 ≤ 7100 * 8

   This constraint ensures that the total production of COLI and PEPSA does not exceed the daily production capacity of

3. PEPSA production constraint:
   X2 ≥ 0.5 * X1

   This constraint ensures that the production of PEPSA is at least half the amount of COLI, as per the firm's commitmen

4. Non-negativity constraints:
   X1 ≥ 0
   X2 ≥ 0

   These constraints ensure that the decision variables X1 and X2 are non-negative, as it is not possible to produce a n

In summary, the constraints for this optimization problem are:

1. X1 / 40 + X2 / 20 ≤ Syrup_Capacity
2. X1 + X2 ≤ 56800
3. X2 ≥ 0.5 * X1
4. X1 ≥ 0
5. 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.

```
message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": (response_p11 + 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 objective function
    model.obj = Objective(expr=49000 * model.X1 - model.X1**2 + 30 * model.X2 - 2 * model.X2**2 - 3.75 * model.X1 - 7.5 * mo

    # Define constraints
    model.syrup_constraint = Constraint(expr=model.X1 / 40 + model.X2 / 20 <= 1000)  # Sample data: Syrup_Capacity = 1000 he
    model.production_constraint = Constraint(expr=model.X1 + model.X2 <= 56800)
    model.pepsa_constraint = Constraint(expr=model.X2 >= 0.5 * model.X1)

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

    # Print the results
    print("Optimal solution:")
    print("X1 (COLI production):", model.X1.value)
    print("X2 (PEPSA production):", model.X2.value)
    print("Maximum daily profit:", model.obj())
    ```

    Explanation:

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

    2. We create a `ConcreteModel` instance named `model` to represent the optimization problem.

    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 expression for the daily profit. The `sense=maxi

    5. We define the constraints using `Constraint()`:
       - The `syrup_constraint` ensures that the production of COLI and PEPSA does not exceed the available syrup capacity.
       - The `production_constraint` ensures that the total production of COLI and PEPSA does not exceed the daily productio
       - The `pepsa_constraint` ensures that the production of PEPSA is at least half the amount of COLI.

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

    7. Finally, we print the optimal solution values for `X1` and `X2` and the maximum daily profit.

    Note: We used sample data of 1000 hectolitres for `Syrup_Capacity`. You can replace this with the actual value if availa

    This code formulates the optimization problem using Pyomo, defines the decision variables, objective function, and const
```

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 69.5 MB/s eta 0:00:00
      ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.6/49.6 kB 5.3 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 objective function
model.obj = Objective(expr=49000 * model.X1 - model.X1**2 + 30 * model.X2 - 2 * model.X2**2 - 3.75 * model.X1 - 7.5 * model.X

# Define constraints
model.syrup_constraint = Constraint(expr=model.X1 / 40 + model.X2 / 20 <= 1000)  # Sample data: Syrup_Capacity = 1000 hectoli
model.production_constraint = Constraint(expr=model.X1 + model.X2 <= 56800)
model.pepsa_constraint = Constraint(expr=model.X2 >= 0.5 * model.X1)

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

# Print the results
print("Optimal solution:")
print("X1 (COLI production):", model.X1.value)
print("X2 (PEPSA production):", model.X2.value)
print("Maximum daily profit:", model.obj())
```
```

```
Optimal solution:
X1 (COLI production): 16335.833333339895
X2 (PEPSA production): 8167.916666659985
Maximum daily profit: 400289176.041992
```

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