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/LP/LP3.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:
 The PRODA, S.A. industrial products firm has to face the problem of scheduling
the weekly production of its three products (P1, P2 and P3). These products are
sold to large industrial firms and PRODA, S.A. wishes to supply its products in
quantities that are more profitable for it.

Each product entails three operations contributing to the costs: smelting; mechanisation; assembly and
packaging. The smelting operations for products P1 and P2 could be subcontracted, but the smelting operation for product
preventing the use of subcontracts. PRODA also want to know, how much they should subcontract.

For product P1 the direct unit costs of all possible operations are:
- smelting at PRODA: 0.30$
- subcontracted smelting: 0.50$
- mechanisation: 0.20$
```

```
- Assembly and packaging: 0.3$
The unit sales price is 1.50$.

For product P2 the direct unit costs of all possible operations are:
- smelting at PRODA: 0.50$
- subcontracted smelting: 0.60$
- mechanisation: 0.10$
- Assembly and packaging: 0.20$
The unit sales price is 1.80$.

For product P3 the direct unit costs of all possible operations are:
- smelting at PRODA: 0.40$
- mechanisation: 0.27$
- Assembly and packaging: 0.20$
The unit sales price is 1.97$.

Each unit of product P1 requires 6 min of smelting time (if performed at PRODA, S.A.), 6 min of mechanisation time and 3
The objective is to maximize weekly profits.
```

```
!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)
```

```
Let's define the variables for this problem:

Let x1 be the number of units of product P1 produced with smelting at PRODA.
Let y1 be the number of units of product P1 produced with subcontracted smelting.
Let x2 be the number of units of product P2 produced with smelting at PRODA.
Let y2 be the number of units of product P2 produced with subcontracted smelting.
Let x3 be the number of units of product P3 produced (all smelting must be done at PRODA).

Therefore, the total number of units produced for each product is:
- Product P1: x1 + y1
- Product P2: x2 + y2
- Product P3: x3
```

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)
```

```
Let's define the objective function for this problem.

The objective is to maximize weekly profits. The profit for each product is the difference between the sales price and t

For product P1:
- Profit per unit when smelting at PRODA = 1.50 - (0.30 + 0.20 + 0.30) = 0.70
- Profit per unit when smelting is subcontracted = 1.50 - (0.50 + 0.20 + 0.30) = 0.50

For product P2:
- Profit per unit when smelting at PRODA = 1.80 - (0.50 + 0.10 + 0.20) = 1.00
- Profit per unit when smelting is subcontracted = 1.80 - (0.60 + 0.10 + 0.20) = 0.90

For product P3:
- Profit per unit = 1.97 - (0.40 + 0.27 + 0.20) = 1.10

The objective function is the sum of the profits from each product:

Maximize: 0.70x1 + 0.50y1 + 1.00x2 + 0.90y2 + 1.10x3

This objective function represents the total weekly profit that PRODA, S.A. aims to maximize by determining the optimal
```

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)
```

```
Now, let's formulate the constraints for this problem:

1. Smelting capacity constraint:
   6x1 + 10x2 + 8x3 ≤ 8000
   (The total time used for in-house smelting cannot exceed the available 8,000 minutes per week)

2. Mechanisation capacity constraint:
   6(x1 + y1) + 3(x2 + y2) + 8x3 ≤ 12000
   (The total time used for mechanisation cannot exceed the available 12,000 minutes per week)

3. Assembly and packaging capacity constraint:
   3(x1 + y1) + 2(x2 + y2) + 2x3 ≤ 10000
   (The total time used for assembly and packaging cannot exceed the available 10,000 minutes per week)

4. Non-negativity constraints:
   x1, y1, x2, y2, x3 ≥ 0
   (The production quantities cannot be negative)

These constraints ensure that the production quantities respect the available capacities for smelting, mechanisation, an
```

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 Pyomo code for the optimization problem:

```python
from pyomo.environ import *

# Create a Concrete Model
model = ConcreteModel()

# Define decision variables
model.x1 = Var(domain=NonNegativeReals)
model.y1 = Var(domain=NonNegativeReals)
model.x2 = Var(domain=NonNegativeReals)
model.y2 = Var(domain=NonNegativeReals)
model.x3 = Var(domain=NonNegativeReals)

# Define the objective function
model.profit = Objective(expr=0.70*model.x1 + 0.50*model.y1 + 1.00*model.x2 + 0.90*model.y2 + 1.10*model.x3, sense=max

# Define constraints
model.smelting_capacity = Constraint(expr=6*model.x1 + 10*model.x2 + 8*model.x3 <= 8000)
model.mechanisation_capacity = Constraint(expr=6*(model.x1 + model.y1) + 3*(model.x2 + model.y2) + 8*model.x3 <= 12000
model.assembly_packaging_capacity = Constraint(expr=3*(model.x1 + model.y1) + 2*(model.x2 + model.y2) + 2*model.x3 <=

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

# Print the results
print("Optimization Results:")
print("x1 (P1 with in-house smelting):", model.x1.value)
print("y1 (P1 with subcontracted smelting):", model.y1.value)
print("x2 (P2 with in-house smelting):", model.x2.value)
print("y2 (P2 with subcontracted smelting):", model.y2.value)
print("x3 (P3 with in-house smelting):", model.x3.value)
print("Total Profit:", model.profit.expr())
```

In this code:

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

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

3. We define the decision variables `x1`, `y1`, `x2`, `y2`, and `x3` as non-negative real variables using `Var(domain=

4. We define the objective function `profit` using the `Objective` component, specifying the expression and the sense

5. We define the constraints using the `Constraint` component, specifying the constraint expressions.

6. We create an instance of the solver (`glpk` in this case) using `SolverFactory`.

7. We solve the optimization problem using `solver.solve(model)` and store the results in the `results` variable.

8. Finally, we print the optimal values of the decision variables and the total profit.

Note: This code assumes that you have the GLPK solver installed. If you want to use a different solver, you can replac

This code provides a complete implementation of the optimization problem using Pyomo in Python. The sample data is use
```

4. Input Problem Data and Test Model Code

```
# Installing pyomo nd solver
!pip install -q pyomo
!pip install pandas
!apt-get install -y -qq glpk-utils
!pip install glpk
```

Show hidden output

```
from pyomo.environ import *
```

```
# Create a Concrete Model
model = ConcreteModel()

# Define decision variables
model.x1 = Var(domain=NonNegativeReals)
model.y1 = Var(domain=NonNegativeReals)
model.x2 = Var(domain=NonNegativeReals)
model.y2 = Var(domain=NonNegativeReals)
model.x3 = Var(domain=NonNegativeReals)

# Define the objective function
model.profit = Objective(expr=0.70*model.x1 + 0.50*model.y1 + 1.00*model.x2 + 0.90*model.y2 + 1.10*model.x3, sense=maximize)

# Define constraints
model.smelting_capacity = Constraint(expr=6*model.x1 + 10*model.x2 + 8*model.x3 <= 8000)
model.mechanisation_capacity = Constraint(expr=6*(model.x1 + model.y1) + 3*(model.x2 + model.y2) + 8*model.x3 <= 12000)
model.assembly_packaging_capacity = Constraint(expr=3*(model.x1 + model.y1) + 2*(model.x2 + model.y2) + 2*model.x3 <= 10000)

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

# Print the results
print("Optimization Results:")
print("x1 (P1 with in-house smelting):", model.x1.value)
print("y1 (P1 with subcontracted smelting):", model.y1.value)
print("x2 (P2 with in-house smelting):", model.x2.value)
print("y2 (P2 with subcontracted smelting):", model.y2.value)
print("x3 (P3 with in-house smelting):", model.x3.value)
print("Total Profit:", model.profit.expr())
```

```
Optimization Results:
x1 (P1 with in-house smelting): 0.0
y1 (P1 with subcontracted smelting): 0.0
x2 (P2 with in-house smelting): 800.0
y2 (P2 with subcontracted smelting): 3200.0
x3 (P3 with in-house smelting): 0.0
Total Profit: 3680.0
```

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