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/IP/IP3.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 the person in charge of packing in a large company. Your job is to skillfully pack items of various weights in
```

```
!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 variables for this optimization problem, let's define:

1. Sets:
   - I: Set of items, indexed by i
   - J: Set of boxes, indexed by j

2. Parameters:
   - w[i]: Weight of item i
   - C: Capacity of each box
   - U: Upper bound on the number of boxes

3. Decision variables:
   - x[i,j]: Binary variable; 1 if item i is packed into box j, 0 otherwise
   - y[j]: Binary variable; 1 if box j is used, 0 otherwise

These variables will be used to formulate the objective function and constraints of the optimization model.
```

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)
```

```
Thank you for providing the sets, parameters, and decision variables. Now, let's formulate the objective function and co

Objective function:
- Minimize the total number of boxes used
  minimize ∑[j ∈ J] y[j]

Constraints:
1. Each item must be packed into exactly one box:
   ∑[j ∈ J] x[i,j] = 1, ∀i ∈ I

2. The total weight of items packed into each box must not exceed the box capacity:
   ∑[i ∈ I] w[i] * x[i,j] ≤ C * y[j], ∀j ∈ J

3. The number of boxes used must not exceed the upper bound:
   ∑[j ∈ J] y[j] ≤ U

4. Binary restrictions on the decision variables:
   x[i,j] ∈ {0, 1}, ∀i ∈ I, ∀j ∈ J
   y[j] ∈ {0, 1}, ∀j ∈ J

The complete mathematical optimization model for the packing problem is as follows:
minimize ∑[j ∈ J] y[j]

subject to:
1. ∑[j ∈ J] x[i,j] = 1, ∀i ∈ I
2. ∑[i ∈ I] w[i] * x[i,j] ≤ C * y[j], ∀j ∈ J
3. ∑[j ∈ J] y[j] ≤ U
4. x[i,j] ∈ {0, 1}, ∀i ∈ I, ∀j ∈ J
   y[j] ∈ {0, 1}, ∀j ∈ J

This model aims to minimize the total number of boxes used while ensuring that each item is packed into exactly one box,
```

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)
```

```
Here are the constraints for the packing optimization problem:

1. Each item must be packed into exactly one box:
   ∑[j ∈ J] x[i,j] = 1, ∀i ∈ I

2. The total weight of items packed into each box must not exceed the box capacity:
   ∑[i ∈ I] w[i] * x[i,j] ≤ C * y[j], ∀j ∈ J

3. The number of boxes used must not exceed the upper bound:
   ∑[j ∈ J] y[j] ≤ U

4. Binary restrictions on the decision variables:
   x[i,j] ∈ {0, 1}, ∀i ∈ I, ∀j ∈ J
   y[j] ∈ {0, 1}, ∀j ∈ J
```

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_p12_adjusted = """
Thank you for providing the sets, parameters, and decision variables. Now, let's formulate the objective function and constr

Objective function:
- Minimize the total number of boxes used
  minimize ∑[j ∈ J] y[j]

The complete mathematical optimization model for the packing problem is as follows:

minimize ∑[j ∈ J] y[j]

This model aims to minimize the total number of boxes used
"""
```

```
message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": (response_p11 + response_p12_adjusted + response_p13)}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

```
def solve_packing_problem(weights, capacity, max_boxes):
    model = ConcreteModel()

    # Sets
    model.I = RangeSet(1, len(weights))
    model.J = RangeSet(1, max_boxes)

    # Parameters
    model.w = Param(model.I, initialize=dict(zip(model.I, weights)))
    model.C = Param(initialize=capacity)
    model.U = Param(initialize=max_boxes)

    # Decision variables
    model.x = Var(model.I, model.J, domain=Binary)
    model.y = Var(model.J, domain=Binary)

    # Objective function
    model.obj = Objective(expr=sum(model.y[j] for j in model.J), sense=minimize)

    # Constraints
    model.item_packing = ConstraintList()
    for i in model.I:
        model.item_packing.add(sum(model.x[i, j] for j in model.J) == 1)

    model.capacity = ConstraintList()
    for j in model.J:
        model.capacity.add(sum(model.w[i] * model.x[i, j] for i in model.I) <= model.C * model.y[j])

    model.box_limit = Constraint(expr=sum(model.y[j] for j in model.J) <= model.U)

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

    # Print the results
    print("Optimal solution:")
    for j in model.J:
        if value(model.y[j]) == 1:
            print(f"Box {j} is used.")
            for i in model.I:
                if value(model.x[i, j]) == 1:
                    print(f"  Item {i} is packed into Box {j}.")

# Sample data
weights = [5, 3, 7, 2, 4]  # Weights of items
capacity = 10  # Capacity of each box
max_boxes = 3  # Upper bound on the number of boxes

solve_packing_problem(weights, capacity, max_boxes)
```

In this code, we define a function `solve_packing_problem` that takes the weights of items, capacity of each box, and

The sample data is provided at the end of the code, where we have a list of item weights (`weights`), the capacity of

Finally, we solve the optimization problem using the 'glpk' solver and print the optimal solution, indicating which bo

Note: Make sure you have Pyomo and the 'glpk' solver installed to run this code successfully.
```
```

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 *
```

```
def solve_packing_problem(weights, capacity, max_boxes):
    model = ConcreteModel()

    # Sets
    model.I = RangeSet(1, len(weights))
    model.J = RangeSet(1, max_boxes)

    # Parameters
    model.w = Param(model.I, initialize=dict(zip(model.I, weights)))
    model.C = Param(initialize=capacity)
    model.U = Param(initialize=max_boxes)

    # Decision variables
    model.x = Var(model.I, model.J, domain=Binary)
    model.y = Var(model.J, domain=Binary)

    # Objective function
    model.obj = Objective(expr=sum(model.y[j] for j in model.J), sense=minimize)

    # Constraints
    model.item_packing = ConstraintList()
    for i in model.I:
        model.item_packing.add(sum(model.x[i, j] for j in model.J) == 1)

    model.capacity = ConstraintList()
    for j in model.J:
        model.capacity.add(sum(model.w[i] * model.x[i, j] for i in model.I) <= model.C * model.y[j])

    model.box_limit = Constraint(expr=sum(model.y[j] for j in model.J) <= model.U)

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

    # Print the results
    print("Optimal solution:")
    for j in model.J:
        if value(model.y[j]) == 1:
            print(f"Box {j} is used.")
            for i in model.I:
                if value(model.x[i, j]) == 1:
                    print(f"  Item {i} is packed into Box {j}.")



    ############ ADDED FOR INTERPRETATION OF RESULTS ############
    bins = {1: [], 2: [], 3: [], 4: [], 5: [], 6: [], 7: [], 8: [], 9: [], 10: [], 11: [], 12: [], 13: []}
    for i in range (1, 25):
      for j in range(1, 14):
        if value(model.x[i,j])> .5:
          bins[j].append(model.w[i])

    print("Bin Division:", bins)

# Sample data
weights = [2, 2, 2, 2, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 6, 6, 7, 7, 8, 8]  # Weights of items
capacity = 9  # Capacity of each box
max_boxes = 13  # Upper bound on the number of boxes

solve_packing_problem(weights, capacity, max_boxes)
```

```
Optimal solution:
Box 1 is used.
  Item 24 is packed into Box 1.
Box 2 is used.
  Item 1 is packed into Box 2.
  Item 21 is packed into Box 2.
Box 3 is used.
  Item 22 is packed into Box 3.
Box 4 is used.
  Item 7 is packed into Box 4.
  Item 18 is packed into Box 4.
Box 5 is used.
  Item 20 is packed into Box 5.
Box 6 is used.
  Item 8 is packed into Box 6.
  Item 14 is packed into Box 6.
Box 7 is used.
  Item 10 is packed into Box 7.
  Item 13 is packed into Box 7.
Box 8 is used.
  Item 9 is packed into Box 8.
  Item 16 is packed into Box 8.
Box 9 is used.
```

```
  Item 5 is packed into Box 9.
  Item 19 is packed into Box 9.
Box 10 is used.
  Item 11 is packed into Box 10.
  Item 17 is packed into Box 10.
Box 11 is used.
  Item 12 is packed into Box 11.
  Item 15 is packed into Box 11.
Box 12 is used.
  Item 23 is packed into Box 12.
Box 13 is used.
  Item 2 is packed into Box 13.
  Item 3 is packed into Box 13.
  Item 4 is packed into Box 13.
  Item 6 is packed into Box 13.
Bin Division: {1: [8], 2: [2, 7], 3: [7], 4: [4, 5], 5: [6], 6: [4, 5], 7: [4, 5], 8: [4, 5], 9: [3, 6], 10: [4, 5], 11:
```

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