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/MIP/MIP4.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 task with maximizing the water flow in a network of pipes over 10 time steps. The water enters the system at po

These are the pipe capacities:
AB: 3
AC: 6
AF: 1
BC: 3
BD: 10
BE: 4
CE: 4
CF: 4
DE: 5
EG: 5
```

```
FB: 12
FG: 7
```

```
!pip install anthropic
```

```
Collecting anthropic
  Downloading anthropic-0.28.0-py3-none-any.whl (862 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 862.7/862.7 kB 13.8 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 9.5 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 23.7 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 7.0 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 7.4 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)
```

```
Here are the variables for the mathematical optimization problem:

Decision variables:
- Let x_ij be a binary variable indicating whether the pipe connecting points i and j is activated (x_ij = 1) or deactiv

- Let f_ij(t) be a continuous variable representing the flow of water through the pipe connecting points i and j at time

- Let r_i(t) be a continuous variable representing the amount of water released out of the system at point i and time st

Input parameters:
- Let c_ij be the capacity of the pipe connecting points i and j, for all available pipes (i,j).

- Let f(t) = max(-4/30t^3 + t^2 - 0.234t + 3, 0) be the function representing the amount of water entering point A at ti
```

```
The goal is to maximize the total amount of water flowing to point G over the 10 time steps, while satisfying the pipe a
```

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)
```

```
Here is the objective function for the given mathematical optimization problem:

Maximize:
  sum(f_FG(t) + f_EG(t) for t = 1 to 10)

This objective function aims to maximize the total amount of water flowing into point G over the 10 time steps. The wate

The full optimization problem would also include the following constraints:
1. Pipe activation constraints: sum(x_ij for all pipes (i,j)) = 10
2. Flow conservation constraints at each point and time step
3. Pipe capacity constraints: f_ij(t) ≤ c_ij * x_ij for all pipes (i,j) and time steps t
4. Non-negativity constraints for flow and release variables

But as requested, I have only formulated the objective function here. Let me know if you need any clarification or have
```

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 given mathematical optimization problem:

1. Pipe activation constraint:
   sum(x_ij for all pipes (i,j)) = 10

2. Flow conservation constraints at each point and time step:
   - Point A: f(t) = f_AB(t) + f_AC(t) + f_AF(t) + r_A(t), for t = 1, 2, ..., 10
   - Point B: f_AB(t-1) + f_FB(t-1) = f_BC(t) + f_BD(t) + f_BE(t) + r_B(t), for t = 2, 3, ..., 10
   - Point C: f_AC(t-1) + f_BC(t-1) = f_CE(t) + f_CF(t) + r_C(t), for t = 2, 3, ..., 10
   - Point D: f_BD(t-1) = f_DE(t) + r_D(t), for t = 2, 3, ..., 10
   - Point E: f_BE(t-1) + f_CE(t-1) + f_DE(t-1) = f_EG(t) + r_E(t), for t = 2, 3, ..., 10
   - Point F: f_AF(t-1) + f_CF(t-1) = f_FB(t) + f_FG(t) + r_F(t), for t = 2, 3, ..., 10
   - Point G: f_EG(t-1) + f_FG(t-1) = r_G(t), for t = 2, 3, ..., 10

3. Pipe capacity constraints:
   f_ij(t) ≤ c_ij * x_ij, for all pipes (i,j) and time steps t = 1, 2, ..., 10

4. Non-negativity constraints:
   - f_ij(t) ≥ 0, for all activated pipes (i,j) and time steps t = 1, 2, ..., 10
   - r_i(t) ≥ 0, for all points i in {A, B, C, D, E, F, G} and time steps t = 1, 2, ..., 10

5. Binary constraints:
   x_ij ∈ {0, 1}, for all pipes (i,j)

These constraints ensure that:
1. Exactly 10 pipes are activated.
2. The flow is conserved at each point and time step, considering the inflow, outflow, and release of water.
```

```
3. The flow through each pipe does not exceed its capacity if the pipe is activated, and is 0 if the pipe is deactivated
4. The flow and release variables are non-negative.
5. The pipe activation variables are binary.

Together with the objective function provided earlier, these constraints complete the formulation of the mathematical op
```

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)
```

```
            print(f"  Flow in pipe ({i}, {j}): {pyo.value(model.f[i, j, t])}")
        print()
```
```

In this code:

- The sets, parameters, and decision variables are defined based on the given information.
- The objective function is defined to maximize the total flow into point G over the 10 time steps.

## 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

```
import pyomo.environ as pyo
import math

# Create a Pyomo model
model = pyo.ConcreteModel()

# Define the sets
model.points = pyo.Set(initialize=['A', 'B', 'C', 'D', 'E', 'F', 'G'])
model.pipes = pyo.Set(initialize=[('A', 'B'), ('A', 'C'), ('A', 'F'), ('B', 'C'), ('B', 'D'), ('B', 'E'),
                                  ('C', 'E'), ('C', 'F'), ('D', 'E'), ('E', 'G'), ('F', 'B'), ('F', 'G')])
model.time_steps = pyo.RangeSet(1, 10)

# Define the parameters
pipe_capacities = {('A', 'B'): 3, ('A', 'C'): 6, ('A', 'F'): 1, ('B', 'C'): 3, ('B', 'D'): 10, ('B', 'E'): 4,
                   ('C', 'E'): 4, ('C', 'F'): 4, ('D', 'E'): 5, ('E', 'G'): 5, ('F', 'B'): 12, ('F', 'G'): 7}
model.c = pyo.Param(model.pipes, initialize=pipe_capacities)

def f(model, t):
    return max(-4/30*t**3 + t**2 - 0.234*t + 3, 0)

# Define the decision variables
model.x = pyo.Var(model.pipes, domain=pyo.Binary)
model.f = pyo.Var(model.pipes, model.time_steps, domain=pyo.NonNegativeReals)
model.r = pyo.Var(model.points, model.time_steps, domain=pyo.NonNegativeReals)

# Define the objective function
def objective_rule(model):
    return sum(model.f[('F', 'G'), t] + model.f[('E', 'G'), t] for t in model.time_steps)
model.objective = pyo.Objective(rule=objective_rule, sense=pyo.maximize)

# Define the constraints
def pipe_activation_rule(model):
    return sum(model.x[i, j] for i, j in model.pipes) == 10
model.pipe_activation = pyo.Constraint(rule=pipe_activation_rule)

def flow_conservation_A_rule(model, t):
    return f(model, t) == model.f['A', 'B', t] + model.f['A', 'C', t] + model.f['A', 'F', t] + model.r['A', t]
model.flow_conservation_A = pyo.Constraint(model.time_steps, rule=flow_conservation_A_rule)

def flow_conservation_B_rule(model, t):
    if t == 1:
        return pyo.Constraint.Skip
    return model.f['A', 'B', t-1] + model.f['F', 'B', t-1] == model.f['B', 'C', t] + model.f['B', 'D', t] + model.f['B', 'E',
model.flow_conservation_B = pyo.Constraint(model.time_steps, rule=flow_conservation_B_rule)

def flow_conservation_C_rule(model, t):
    if t == 1:
        return pyo.Constraint.Skip
    return model.f['A', 'C', t-1] + model.f['B', 'C', t-1] == model.f['C', 'E', t] + model.f['C', 'F', t] + model.r['C', t]
model.flow_conservation_C = pyo.Constraint(model.time_steps, rule=flow_conservation_C_rule)

def flow_conservation_D_rule(model, t):
    if t == 1:
        return pyo.Constraint.Skip
    return model.f['B', 'D', t-1] == model.f['D', 'E', t] + model.r['D', t]
model.flow_conservation_D = pyo.Constraint(model.time_steps, rule=flow_conservation_D_rule)

def flow_conservation_E_rule(model, t):
    if t == 1:
        return pyo.Constraint.Skip
```
```

```
    return model.f['B', 'E', t-1] + model.f['C', 'E', t-1] + model.f['D', 'E', t-1] == model.f['E', 'G', t] + model.r['E', t]
model.flow_conservation_E = pyo.Constraint(model.time_steps, rule=flow_conservation_E_rule)

def flow_conservation_F_rule(model, t):
    if t == 1:
        return pyo.Constraint.Skip
    return model.f['A', 'F', t-1] + model.f['C', 'F', t-1] == model.f['F', 'B', t] + model.f['F', 'G', t] + model.r['F', t]
model.flow_conservation_F = pyo.Constraint(model.time_steps, rule=flow_conservation_F_rule)

def flow_conservation_G_rule(model, t):
    if t == 1:
        return pyo.Constraint.Skip
    return model.f['E', 'G', t-1] + model.f['F', 'G', t-1] == model.r['G', t]
model.flow_conservation_G = pyo.Constraint(model.time_steps, rule=flow_conservation_G_rule)

def pipe_capacity_rule(model, i, j, t):
    return model.f[i, j, t] <= model.c[i, j] * model.x[i, j]
model.pipe_capacity = pyo.Constraint(model.pipes, model.time_steps, rule=pipe_capacity_rule)

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

# Print the results
print("Objective value:", pyo.value(model.objective))

print("Activated pipes:")
for i, j in model.pipes:
    if pyo.value(model.x[i, j]) == 1:
        print(f"Pipe ({i}, {j})")

print("Flow values:")
for t in model.time_steps:
    print(f"Time step {t}:")
    for i, j in model.pipes:
        if pyo.value(model.x[i, j]) == 1:
            print(f"  Flow in pipe ({i}, {j}): {pyo.value(model.f[i, j, t])}")
    print()
```

```
Objective value: 81.12266666666669
Activated pipes:
Pipe (A, B)
Pipe (A, C)
Pipe (B, C)
Pipe (B, D)
Pipe (C, E)
Pipe (C, F)
Pipe (D, E)
Pipe (E, G)
Pipe (F, B)
Pipe (F, G)
Flow values:
Time step 1:
  Flow in pipe (A, B): 2.63266666666667
  Flow in pipe (A, C): 1.0
  Flow in pipe (B, C): 3.0
  Flow in pipe (B, D): 5.0
  Flow in pipe (C, E): 4.0
  Flow in pipe (C, F): 4.0
  Flow in pipe (D, E): 1.0
  Flow in pipe (E, G): 5.0
  Flow in pipe (F, B): 3.902
  Flow in pipe (F, G): 7.0

Time step 2:
  Flow in pipe (A, B): 3.0
  Flow in pipe (A, C): 2.46533333333333
  Flow in pipe (B, C): 3.0
  Flow in pipe (B, D): 3.53466666666667
  Flow in pipe (C, E): 0.0
  Flow in pipe (C, F): 4.0
  Flow in pipe (D, E): 5.0
  Flow in pipe (E, G): 5.0
  Flow in pipe (F, B): 0.0
  Flow in pipe (F, G): 4.0

Time step 3:
  Flow in pipe (A, B): 3.0
  Flow in pipe (A, C): 4.698
  Flow in pipe (B, C): 2.0
  Flow in pipe (B, D): 1.0
  Flow in pipe (C, E): 1.46533333333333
  Flow in pipe (C, F): 4.0
  Flow in pipe (D, E): 3.53466666666667
  Flow in pipe (E, G): 5.0
  Flow in pipe (F, B): 0.0
  Flow in pipe (F, G): 4.0
```

```
Time step 4:
  Flow in pipe (A, B): 3.0
  Flow in pipe (A, C): 6.0
  Flow in pipe (B, C): 2.0
  Flow in pipe (B, D): 1.0
  Flow in pipe (C, E): 4.0
  Flow in pipe (C, F): 2.698
  Flow in pipe (D, E): 1.0
  Flow in pipe (E, G): 5.0
```

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