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
prompt1_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt1_MathematicalModel.txt'
prompt2_path = '/content/drive/MyDrive/Thesis/Prompts/Prompt2_PyomoCode.txt'
problem_desc_path = '/content/drive/MyDrive/Thesis/ProblemDescriptions/MIP/MIP4.txt'

prompt1_file = open(prompt1_path, "r")
prompt2_file = open(prompt2_path, "r")
problem_desc_file = open(problem_desc_path, "r")

prompt1 = prompt1_file.read()
print("Prompt 1:\n", prompt1)

prompt2 = prompt2_file.read()
print("Prompt 2:\n", prompt2)

problem_desc = problem_desc_file.read()
print("Problem Description:\n", problem_desc)
```

```
Prompt 1:
 Please write a mathematical optimization model for this problem. Include parameters, decision variables, the objective
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
```

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: Create Mathematical Model

```
message = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt1,
    max_tokens=4096,
    messages=[
        {"role": "user", "content": problem_desc}
    ]
)

response_p1 = message.content[0].text


# Print response
print(response_p1)
```

```
Here's the mathematical optimization model for the given problem:

Parameters:
- T: Total number of time steps (T = 10)
- N: Total number of points in the network (N = 7, points A to G)
- M: Total number of available pipes (M = 12)
- K: Number of pipes that can be activated (K = 10)
- cap[i, j]: Capacity of the pipe connecting points i and j
- f(t): Water inflow at point A at time step t, where f(t) = max(-4/30t^3 + t^2 - 0.234t + 3, 0)

Decision Variables:
- x[i, j, t]: Flow of water from point i to point j at time step t (continuous variable)
- y[i, j]: Binary variable indicating if the pipe connecting points i and j is activated (1 if activated, 0 otherwise)
- r[i, t]: Amount of water released out of the system at point i at time step t (continuous variable)

Objective Function:
Maximize: sum(x['G', 'out', t] for t in range(1, T+1))

Constraints:
1. Water inflow at point A:
   x['in', 'A', t] = f(t) for t in range(1, T+1)

2. Water outflow at point G:
   x['G', 'out', t] <= sum(x[i, 'G', t] for i in ['E', 'F']) for t in range(1, T+1)

3. Flow conservation at each point i (except A and G) and time step t:
   sum(x[j, i, t-1] for j in inflow_points[i]) + r[i, t-1] = sum(x[i, j, t] for j in outflow_points[i]) + r[i, t] for i

4. Capacity constraints for each pipe:
   x[i, j, t] <= cap[i, j] * y[i, j] for i, j in pipes, t in range(1, T+1)

5. Activation constraint:
   sum(y[i, j] for i, j in pipes) <= K

6. Non-negativity constraints:
   x[i, j, t] >= 0 for i, j in pipes, t in range(1, T+1)
   r[i, t] >= 0 for i in range(1, N-1), t in range(1, T+1)

7. Binary constraints:
   y[i, j] in {0, 1} for i, j in pipes

Note:
- inflow_points[i] represents the set of points that have a pipe flowing into point i
- outflow_points[i] represents the set of points that have a pipe flowing out of point i
- pipes is the set of all available pipes represented as (i, j) tuples

This optimization model aims to maximize the total water flow at point G over the given time steps while satisfying the
```

2. Prompt 2: Write Code based on Mathematical Model

```
message2 = client.messages.create(
    model="claude-3-opus-20240229",
    system=prompt2,
    max_tokens=4069,
    messages=[
        {"role": "user", "content": response_p1}
    ]
)

response_p2 = message2.content[0].text


# Print response
print(response_p2)
```

```
Here's the Python code using Pyomo to solve the optimization problem:
```

```
```python
import pyomo.environ as pyo
import numpy as np

# Sample data
T = 10  # Total number of time steps
N = 7   # Total number of points in the network (A to G)
M = 12  # Total number of available pipes
K = 10  # Number of pipes that can be activated

# Sample capacity data for pipes
cap = {
    ('in', 'A'): 10, ('A', 'B'): 8, ('A', 'C'): 6, ('B', 'C'): 5,
    ('B', 'D'): 7, ('B', 'E'): 4, ('C', 'D'): 3, ('C', 'E'): 6,
    ('D', 'E'): 5, ('D', 'F'): 8, ('E', 'F'): 4, ('E', 'G'): 9,
    ('F', 'G'): 7, ('G', 'out'): 10
}

# Water inflow function at point A
def f(t):
    return max(-4/30 * t**3 + t**2 - 0.234 * t + 3, 0)

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

# Define sets
model.T = pyo.RangeSet(1, T)
model.N = pyo.RangeSet(1, N-1)  # Excluding point G
model.pipes = pyo.Set(initialize=cap.keys())

# Define decision variables
model.x = pyo.Var(model.pipes, model.T, domain=pyo.NonNegativeReals)
model.y = pyo.Var(model.pipes, domain=pyo.Binary)
model.r = pyo.Var(model.N, model.T, domain=pyo.NonNegativeReals)

# Define objective function
def obj_rule(model):
    return sum(model.x['G', 'out', t] for t in model.T)
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)

# Define constraints
def inflow_rule(model, t):
    return model.x['in', 'A', t] == f(t)
model.inflow_con = pyo.Constraint(model.T, rule=inflow_rule)

def outflow_rule(model, t):
    return model.x['G', 'out', t] <= sum(model.x[i, 'G', t] for i in ['E', 'F'])
model.outflow_con = pyo.Constraint(model.T, rule=outflow_rule)

def flow_conservation_rule(model, i, t):
    inflow_points = [j for j, k in model.pipes if k == i]
    outflow_points = [k for j, k in model.pipes if j == i]
    if t == 1:
        return sum(model.x[j, i, t] for j in inflow_points) + model.r[i, t] == sum(model.x[i, j, t] for j in outflow_p
    else:
```

4. Input Problem Data and Test Model Code

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

Show hidden output

```
import pyomo.environ as pyo
import numpy as np

# Sample data
T = 10  # Total number of time steps
N = 7   # Total number of points in the network (A to G)
M = 12  # Total number of available pipes
K = 10  # Number of pipes that can be activated

# Sample capacity data for pipes
cap = {
    ('1', '2'): 3, ('1', '3'): 6, ('1', '6'): 1, ('2', '3'): 3,
    ('2', '4'): 10, ('2', '5'): 4, ('3', '5'): 4, ('3', '6'): 4,
    ('4', '5'): 5, ('5', '7'): 5, ('6', '2'): 12, ('6', '7'): 7,
    ('7', 'out'): 1000, ('in', '1'): 1000
}

# Water inflow function at point A
```
```

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

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

# Define sets
model.T = pyo.RangeSet(1, T)
model.N = pyo.RangeSet(1, N-1)  # Excluding point G
model.pipes = pyo.Set(initialize=cap.keys())

# Define decision variables
model.x = pyo.Var(model.pipes, model.T, domain=pyo.NonNegativeReals)
model.y = pyo.Var(model.pipes, domain=pyo.Binary)
model.r = pyo.Var(model.N, model.T, domain=pyo.NonNegativeReals)

# Define objective function
def obj_rule(model):
    return sum(model.x['7', 'out', t] for t in model.T)
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)

# Define constraints
def inflow_rule(model, t):
    return model.x['in', '1', t] == f(t)
model.inflow_con = pyo.Constraint(model.T, rule=inflow_rule)

def outflow_rule(model, t):
    return model.x['7', 'out', t] <= sum(model.x[i, '7', t] for i in ['5', '6'])
model.outflow_con = pyo.Constraint(model.T, rule=outflow_rule)

def flow_conservation_rule(model, i, t): # MODIFIED TO USE str(i) INSTEAD OF i SO CODE WORKS (data loading issue)
    inflow_points = [j for j, k in model.pipes if k == str(i)]
    outflow_points = [k for j, k in model.pipes if j == str(i)]
    if t == 1:
        return sum(model.x[j, str(i), t] for j in inflow_points) + model.r[i, t] == sum(model.x[str(i), j, t] for j in outflo
    else:
        return sum(model.x[j, str(i), t-1] for j in inflow_points) + model.r[i, t-1] == sum(model.x[str(i), j, t] for j in ou
model.flow_conservation_con = pyo.Constraint(model.N, model.T, rule=flow_conservation_rule)

def capacity_rule(model, i, j, t):
    return model.x[i, j, t] <= cap[i, j] * model.y[i, j]
model.capacity_con = pyo.Constraint(model.pipes, model.T, rule=capacity_rule)

def activation_rule(model):
    return sum(model.y[i, j] for i, j in model.pipes) <= K
model.activation_con = pyo.Constraint(rule=activation_rule)

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

# Print the results
print("Objective value:", pyo.value(model.obj))
for i, j in model.pipes:
    if pyo.value(model.y[i, j]) > 0:
        print(f"Pipe ({i}, {j}) is activated")
        for t in model.T:
            print(f"Flow from {i} to {j} at time {t}: {pyo.value(model.x[i, j, t])}")
```

```
Objective value: 91.16333333333333
Pipe (1, 2) is activated
Flow from 1 to 2 at time 1: 3.0
Flow from 1 to 2 at time 2: 3.0
Flow from 1 to 2 at time 3: 1.0
Flow from 1 to 2 at time 4: 1.698
Flow from 1 to 2 at time 5: 3.0
Flow from 1 to 2 at time 6: 3.0
Flow from 1 to 2 at time 7: 3.0
Flow from 1 to 2 at time 8: 0.118666666666664
Flow from 1 to 2 at time 9: 0.0
Flow from 1 to 2 at time 10: 0.0
Pipe (1, 3) is activated
Flow from 1 to 3 at time 1: 4.0
Flow from 1 to 3 at time 2: 4.0
Flow from 1 to 3 at time 3: 4.46533333333333
Flow from 1 to 3 at time 4: 6.0
Flow from 1 to 3 at time 5: 6.0
Flow from 1 to 3 at time 6: 6.0
Flow from 1 to 3 at time 7: 6.0
Flow from 1 to 3 at time 8: 6.0
Flow from 1 to 3 at time 9: 0.0
Flow from 1 to 3 at time 10: 0.0
Pipe (2, 4) is activated
Flow from 2 to 4 at time 1: 5.0
```

```
Flow from 2 to 4 at time 2: 5.0
Flow from 2 to 4 at time 3: 3.0
Flow from 2 to 4 at time 4: 1.0
Flow from 2 to 4 at time 5: 1.698
Flow from 2 to 4 at time 6: 3.0
Flow from 2 to 4 at time 7: 3.0
Flow from 2 to 4 at time 8: 3.0
Flow from 2 to 4 at time 9: 0.118666666666664
Flow from 2 to 4 at time 10: 0.0
Pipe (3, 5) is activated
Flow from 3 to 5 at time 1: 0.0
Flow from 3 to 5 at time 2: -4.44089209850063e-16
Flow from 3 to 5 at time 3: 0.0
Flow from 3 to 5 at time 4: 2.0
Flow from 3 to 5 at time 5: 4.0
Flow from 3 to 5 at time 6: 2.0
Flow from 3 to 5 at time 7: 2.0
Flow from 3 to 5 at time 8: 2.0
Flow from 3 to 5 at time 9: 2.0
Flow from 3 to 5 at time 10: 0.0
Pipe (3, 6) is activated
Flow from 3 to 6 at time 1: 4.0
Flow from 3 to 6 at time 2: 4.0
Flow from 3 to 6 at time 3: 4.0
Flow from 3 to 6 at time 4: 0.465333333333334
Flow from 3 to 6 at time 5: 4.0
Flow from 3 to 6 at time 6: 4.0
Flow from 3 to 6 at time 7: 4.0
Flow from 3 to 6 at time 8: 4.0
Flow from 3 to 6 at time 9: 4.0
Flow from 3 to 6 at time 10: 0.0
Pipe (4, 5) is activated
```

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