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)
```

```
Based on the given information, we can formulate a mathematical optimization model as follows:

Parameters:
- T: Total number of time steps (T = 10)
- N: Total number of points in the network (N = 7, labeled A to G)
- P: Total number of available pipes (P = 12)
- K: Maximum number of pipes that can be activated (K = 10)
- C[i, j]: Capacity of the pipe connecting points i and j (given in the problem statement)
- 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]: Amount of water flowing from point i to point j at time step t (continuous variable)
- y[i, j]: Binary variable indicating whether the pipe connecting points i and j is activated (1 if activated, 0 otherwi
- 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['F', 'G', t] for t in range(1, T+1)) + sum(x['E', 'G', t] for t in range(1, T+1))

Constraints:
1. Flow conservation at each point i and time step t (except for point A at t=1 and point G):
   sum(x[j, i, t] for j in range(1, N+1) if (j, i) is a valid pipe) + r[i, t] = sum(x[i, j, t+1] for j in range(1, N+1)

2. Inflow at point A at time step t=1:
   sum(x[A, j, 1] for j in range(1, N+1) if (A, j) is a valid pipe) = f(1)

3. Capacity constraints for activated pipes:
   x[i, j, t] <= C[i, j] * y[i, j] for all valid pipes (i, j) and t in range(1, T+1)

4. Pipe activation constraint:
   sum(y[i, j] for all valid pipes (i, j)) <= K

5. Binary constraints for pipe activation variables:
   y[i, j] is binary for all valid pipes (i, j)

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

The objective function aims to maximize the total amount of water flowing into point G over all time steps. The constrai
```

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 a Python code using Pyomo to solve the optimization problem:

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

# Define the model
model = pyo.ConcreteModel()

# Define the parameters
````
````

```
    T = 10
    N = 7
    P = 12
    K = 10

    # Sample data for pipe capacities (replace with actual data)
    C = {
        ('A', 'B'): 5, ('A', 'C'): 7, ('A', 'D'): 3,
        ('B', 'C'): 4, ('B', 'E'): 6,
        ('C', 'D'): 2, ('C', 'E'): 5, ('C', 'F'): 8,
        ('D', 'F'): 4,
        ('E', 'F'): 3, ('E', 'G'): 6,
        ('F', 'G'): 5
    }

    # Define the water inflow function
    def f(t):
        return max(-4/30*t**3 + t**2 - 0.234*t + 3, 0)

    # Define the sets
    model.T = pyo.RangeSet(1, T)
    model.N = pyo.RangeSet(1, N)
    model.P = pyo.Set(initialize=C.keys())

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

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

    # Define the constraints
    def flow_conservation_rule(model, i, t):
        if i == 1 or i == N:
            return pyo.Constraint.Skip
        inflow = sum(model.x[j, i, t] for j in model.N if (j, i) in model.P)
        outflow = sum(model.x[i, j, t+1] for j in model.N if (i, j) in model.P)
        return inflow + model.r[i, t] == outflow
    model.flow_conservation = pyo.Constraint(model.N, model.T, rule=flow_conservation_rule)

    def inflow_rule(model):
        return sum(model.x['A', j, 1] for j in model.N if ('A', j) in model.P) == f(1)
    model.inflow = pyo.Constraint(rule=inflow_rule)
```

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

# Define the model
model = pyo.ConcreteModel()

# Define the parameters
T = 10
N = 7
P = 12
K = 10

# Sample data for pipe capacities (replace with actual data)
C = {
    (0, 1): 3, (0, 2): 6, (0, 5): 1, (1, 2): 3,
    (1, 3): 10, (1, 4): 4, (2, 4): 4, (2, 5): 4,
    (3, 4): 5, (4, 6): 5, (5, 1): 12, (5, 6): 7
}

# Define the water inflow function
def f(t):
    return max(-4/30*t**3 + t**2 - 0.234*t + 3, 0)

# Define the sets
model.T = pyo.RangeSet(1, T)
model.N = pyo.RangeSet(1, N)
model.P = pyo.Set(initialize=C.keys())

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

# Define the objective function
def obj_rule(model):
    return sum(model.x[5, 6, t] + model.x[4, 6, t] for t in model.T)
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)

# Define the constraints
def flow_conservation_rule(model, i, t):
    if i == 1 or i == N:
        return pyo.Constraint.Skip
    inflow = sum(model.x[j, i, t] for j in model.N if (j, i) in model.P)
    outflow = sum(model.x[i, j, t+1] for j in model.N if (i, j) in model.P)
    return inflow + model.r[i, t] == outflow
model.flow_conservation = pyo.Constraint(model.N, model.T, rule=flow_conservation_rule)

def inflow_rule(model):
    return sum(model.x[1, j, 1] for j in model.N if (1, j) in model.P) == f(1)
model.inflow = pyo.Constraint(rule=inflow_rule)

def capacity_rule(model, i, j, t):
    return model.x[i, j, t] <= C[i, j] * model.y[i, j]
model.capacity = pyo.Constraint(model.P, model.T, rule=capacity_rule)

def pipe_activation_rule(model):
    return sum(model.y[i, j] for (i, j) in model.P) <= K
model.pipe_activation = pyo.Constraint(rule=pipe_activation_rule)

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

# Print the results
print("Objective value:", pyo.value(model.obj))
for (i, j) in model.P:
    if pyo.value(model.y[i, j]) > 0.5:
        print(f"Pipe {i}-{j} activated")
        for t in model.T:
            print(f"  Time {t}: Flow = {pyo.value(model.x[i, j, t])}")
```

```
ERROR:pyomo.core:Rule failed when generating expression for Constraint flow_conservation with index (2, 10):
KeyError: "Index '(2, 4, 11)' is not valid for indexed component 'x'"
ERROR:pyomo.core:Constructing component 'flow_conservation' from data=None failed:
    KeyError: "Index '(2, 4, 11)' is not valid for indexed component 'x'"
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-12-0909a747c7e8> in <cell line: 46>()
     44     outflow = sum(model.x[i, j, t+1] for j in model.N if (i, j) in model.P)
     45     return inflow + model.r[i, t] == outflow
---> 46 model.flow_conservation = pyo.Constraint(model.N, model.T, rule=flow_conservation_rule)
     47 
     48 def inflow_rule(model):

                              8 frames
/usr/local/lib/python3.10/dist-packages/pyomo/core/base/indexed_component.py in _validate_index(self, idx)
    864         # Raise an exception
    865         #
--> 866         raise KeyError(
    867             "Index '%s' is not valid for indexed component '%s'"
    868             % (normalized_idx, self.name)

KeyError: "Index '(2, 4, 11)' is not valid for indexed component 'x'"
```

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

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

# Define the model
model = pyo.ConcreteModel()

# Define the parameters
T = 10
N = 7
P = 12
K = 10

# Sample data for pipe capacities (replace with actual data)
C = {
    (0, 1): 3, (0, 2): 6, (0, 5): 1, (1, 2): 3,
    (1, 3): 10, (1, 4): 4, (2, 4): 4, (2, 5): 4,
    (3, 4): 5, (4, 6): 5, (5, 1): 12, (5, 6): 7
}

# Define the water inflow function
def f(t):
    return max(-4/30*t**3 + t**2 - 0.234*t + 3, 0)

# Define the sets
model.T = pyo.RangeSet(1, T)
model.N = pyo.RangeSet(1, N)
model.P = pyo.Set(initialize=C.keys())

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

# Define the objective function
def obj_rule(model):
    return sum(model.x[5, 6, t] + model.x[4, 6, t] for t in model.T)
model.obj = pyo.Objective(rule=obj_rule, sense=pyo.maximize)

# Define the constraints
def flow_conservation_rule(model, i, t):
    if i == 1 or i == N or t == T: # MODIFIED To INCLUDE t == 10 SO CODE IS CONSISTENT WITH MATH MODEL
        return pyo.Constraint.Skip
    inflow = sum(model.x[j, i, t] for j in model.N if (j, i) in model.P)
    outflow = sum(model.x[i, j, t+1] for j in model.N if (i, j) in model.P)
    return inflow + model.r[i, t] == outflow
model.flow_conservation = pyo.Constraint(model.N, model.T, rule=flow_conservation_rule)

def inflow_rule(model):
    return sum(model.x[1, j, 1] for j in model.N if (1, j) in model.P) == f(1)
model.inflow = pyo.Constraint(rule=inflow_rule)

def capacity_rule(model, i, j, t):
    return model.x[i, j, t] <= C[i, j] * model.y[i, j]
model.capacity = pyo.Constraint(model.P, model.T, rule=capacity_rule)
```

```
def pipe_activation_rule(model):
    return sum(model.y[i, j] for (i, j) in model.P) <= K
model.pipe_activation = pyo.Constraint(rule=pipe_activation_rule)

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

# Print the results
print("Objective value:", pyo.value(model.obj))
for (i, j) in model.P:
    if pyo.value(model.y[i, j]) > 0.5:
        print(f"Pipe {i}-{j} activated")
        for t in model.T:
            print(f"  Time {t}: Flow = {pyo.value(model.x[i, j, t])}")
```

```
ERROR:pyomo.common.numeric_types:evaluating object as numeric value: x[5,6,1]
    (object: <class 'pyomo.core.base.var.VarData'>)
No value for uninitialized NumericValue object x[5,6,1]
ERROR:pyomo.common.numeric_types:evaluating object as numeric value: obj
    (object: <class 'pyomo.core.base.objective.ScalarObjective'>)
No value for uninitialized NumericValue object x[5,6,1]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-18d7bd1e2f82> in <cell line: 65>()
```