Universidad de los Andes

Laboratorio 4
Manejo básico de Pyomo

Modelado, Optimización y Simulación

Profesor
Germán Montoya

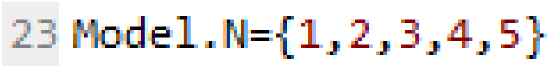
Oficina ML648

Agenda

- Conjuntos
- Parámetros
- Variables
- Ejemplos
  - Función objetivo
  - Restricciones
    - Sumatorias y “para todo”.

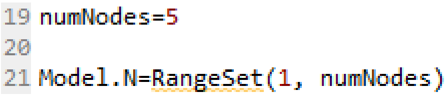
Conjuntos

- Definición básica:



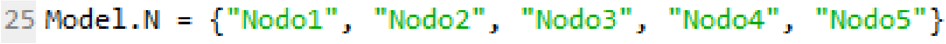
```
Model.N={1,2,3,4,5}
```

- Definición usando la función RangeSet:



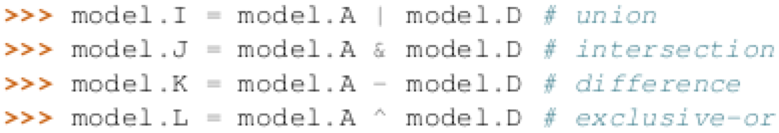
```
numNodes=5

Model.N=RangeSet(1, numNodes)
```

- Definición usando cadenas de caracteres:



```
Model.N = {"Nodo1", "Nodo2", "Nodo3", "Nodo4", "Nodo5"}
```

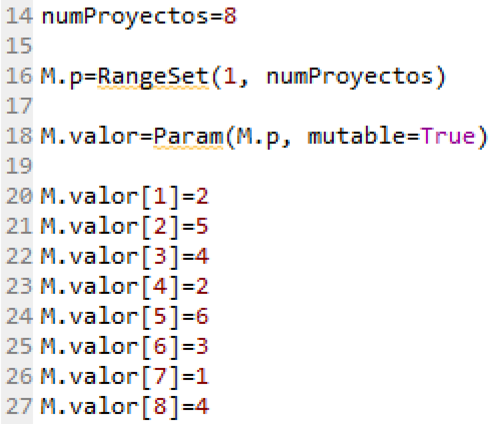
- Operaciones entre conjuntos:



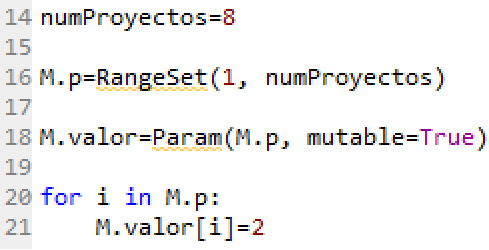
```
>>> model.I = model.A | model.D # union
>>> model.J = model.A & model.D # intersection
>>> model.K = model.A - model.D # difference
>>> model.L = model.A ^ model.D # exclusive-or
```

Parámetros

- Vectores:



```
numProyectos=8

M.p=RangeSet(1, numProyectos)

M.valor=Param(M.p, mutable=True)

M.valor[1]=2
M.valor[2]=5
M.valor[3]=4
M.valor[4]=2
M.valor[5]=6
M.valor[6]=3
M.valor[7]=1
M.valor[8]=4
```

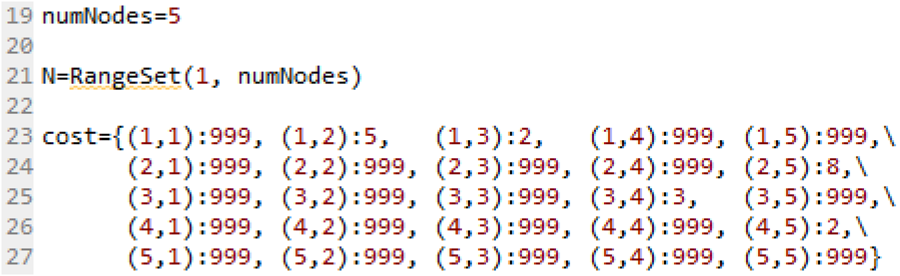


```
numProyectos=8

M.p=RangeSet(1, numProyectos)

M.valor=Param(M.p, mutable=True)

for i in M.p:
    M.valor[i]=2
```

Parámetros

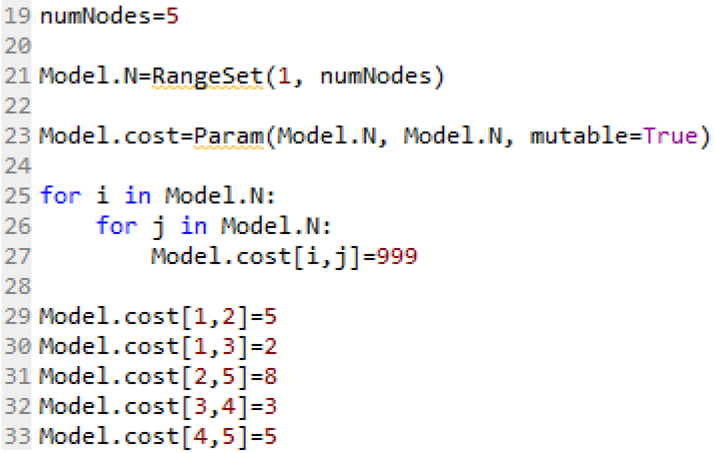
- Matrices:

Forma 1:



```
numNodes=5

N=RangeSet(1, numNodes)

cost={(1,1):999, (1,2):5,   (1,3):2,   (1,4):999, (1,5):999,\
      (2,1):999, (2,2):999, (2,3):999, (2,4):999, (2,5):8,\
      (3,1):999, (3,2):999, (3,3):999, (3,4):3,   (3,5):999,\
      (4,1):999, (4,2):999, (4,3):999, (4,4):999, (4,5):2,\
      (5,1):999, (5,2):999, (5,3):999, (5,4):999, (5,5):999}
```

Forma 2:



```
numNodes=5

Model.N=RangeSet(1, numNodes)

Model.cost=Param(Model.N, Model.N, mutable=True)

for i in Model.N:
    for j in Model.N:
        Model.cost[i,j]=999

Model.cost[1,2]=5
Model.cost[1,3]=2
Model.cost[2,5]=8
Model.cost[3,4]=3
Model.cost[4,5]=5
```

Variables

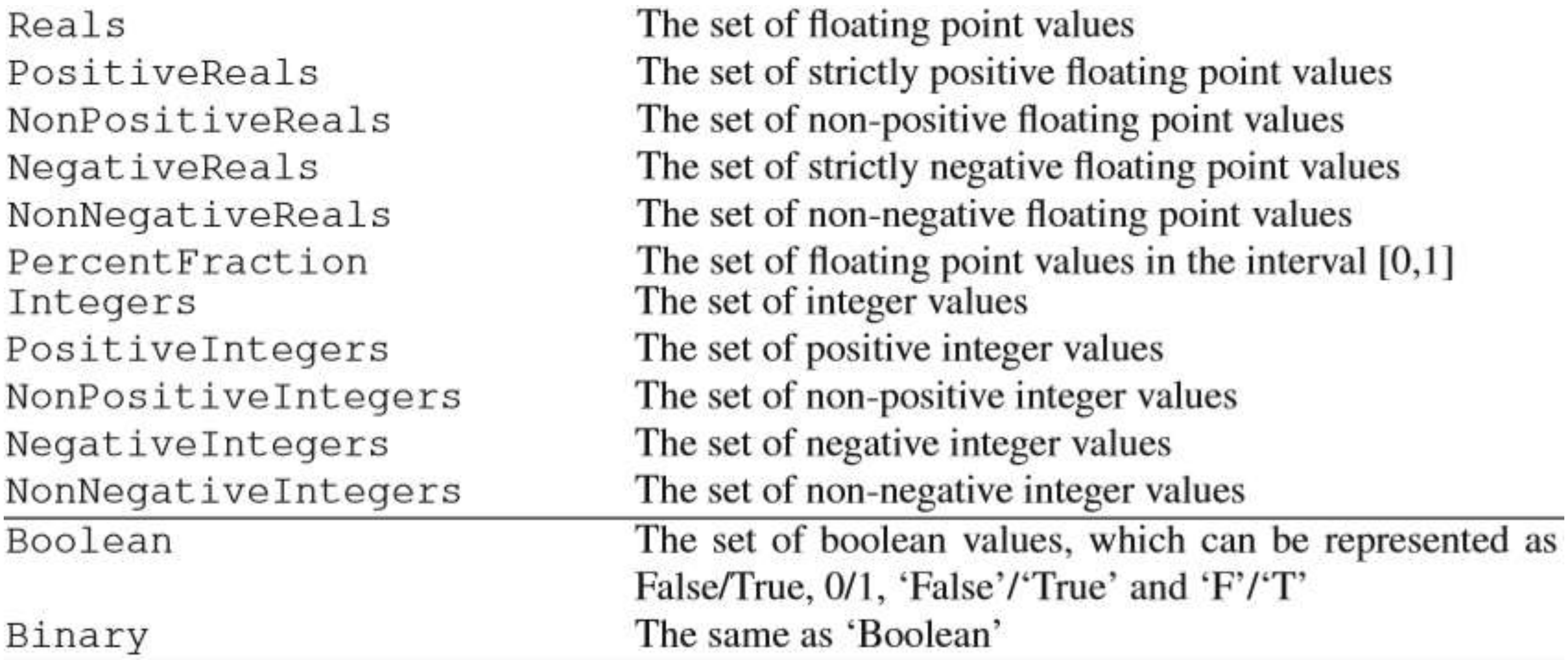
- Variable simple (sin dimensiones):
  - Model.x=Var()

- Dominio de las variables:
  - Forma 1: Model.x=Var(N, domain=NonNegativeReals)
  - Forma 2: Model.x=Var(N, within=NonNegativeReals)

- Limite superior e inferior de las variables:
  - Ej1: Model.x=Var(domain=NonNegativeReals, bounds=(0,6))
  - Ej2: Model.x=Var(N, domain=NonNegativeReals, bounds=(0,6))

Variables

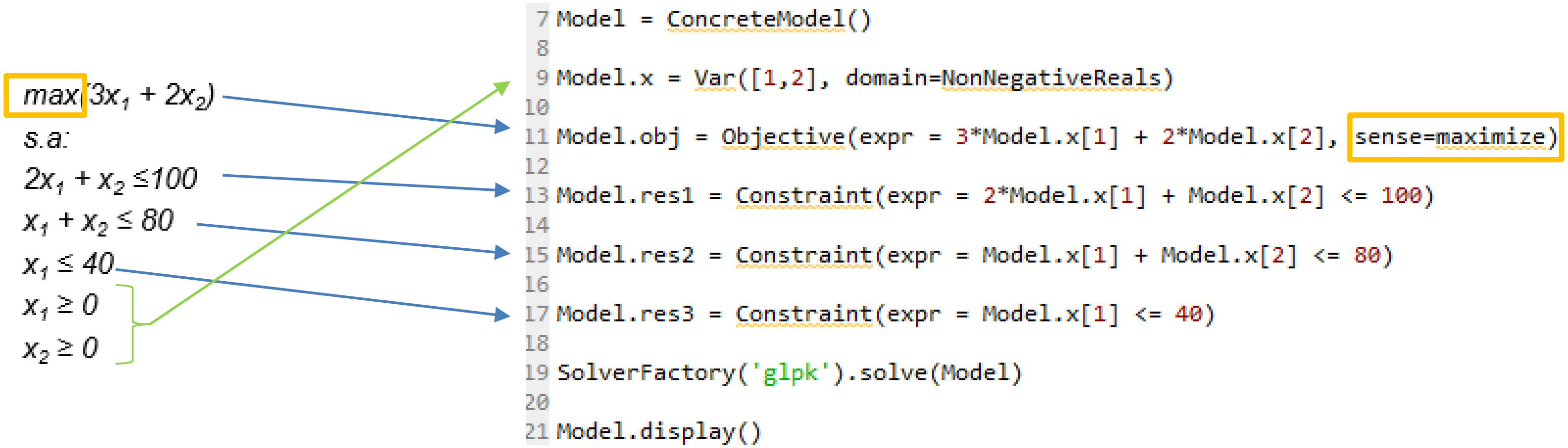
- Dominios posibles para las variables:



Reals	The set of floating point values
PositiveReals	The set of strictly positive floating point values
NonPositiveReals	The set of non-positive floating point values
NegativeReals	The set of strictly negative floating point values
NonNegativeReals	The set of non-negative floating point values
PercentFraction	The set of floating point values in the interval [0,1]
Integers	The set of integer values
PositiveIntegers	The set of positive integer values
NonPositiveIntegers	The set of non-positive integer values
NegativeIntegers	The set of negative integer values
NonNegativeIntegers	The set of non-negative integer values
Boolean	The set of boolean values, which can be represented as False/True, 0/1, 'False'/'True' and 'F'/'T'
Binary	The same as 'Boolean'

Ejemplos

- Caso Woodcarving:



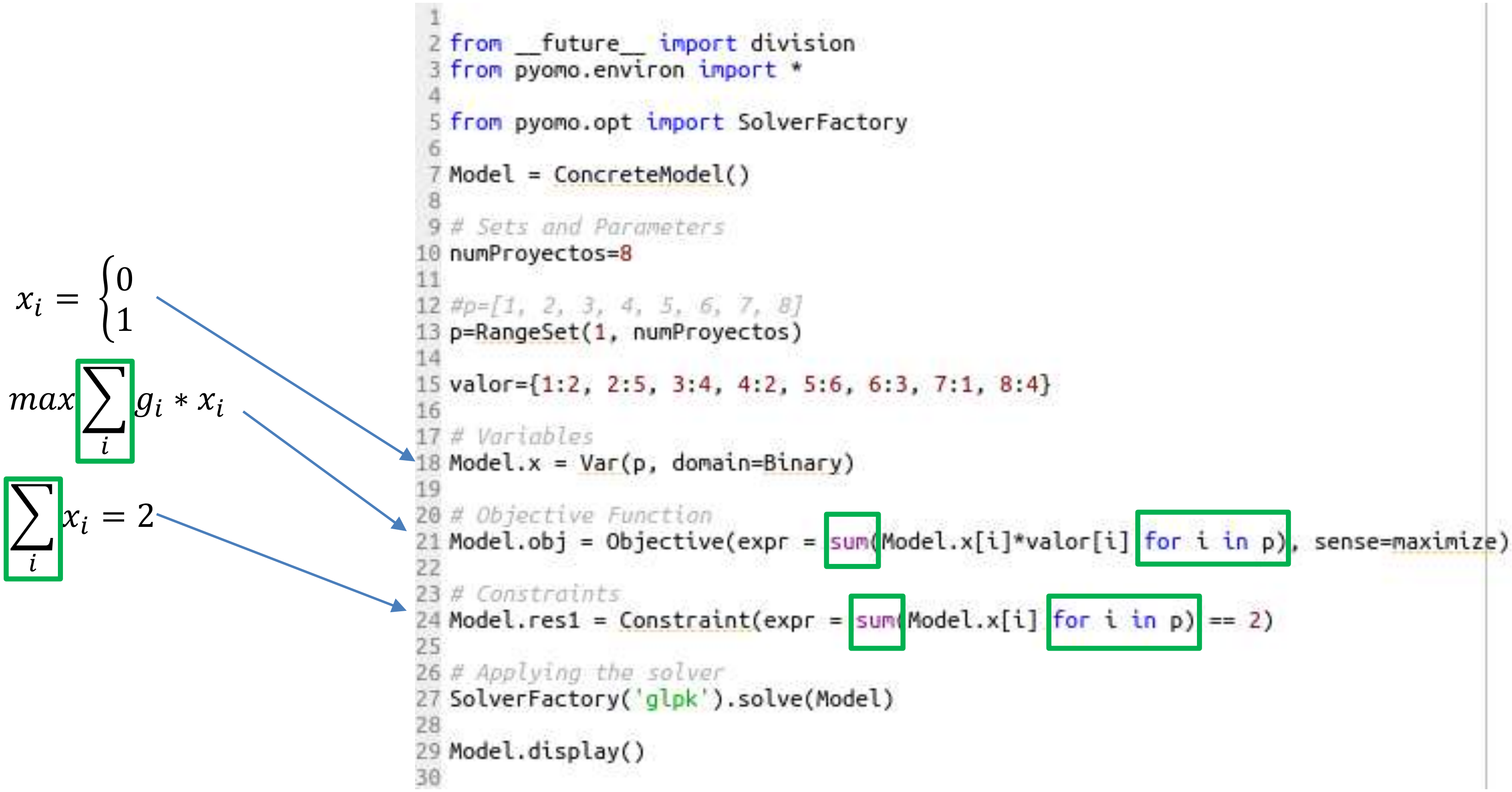
$$
\begin{aligned}
&\boxed{\max}(3x_1 + 2x_2)\\
&s.a:\\
&2x_1 + x_2 \leq 100\\
&x_1 + x_2 \leq 80\\
&x_1 \leq 40\\
&x_1 \geq 0\\
&x_2 \geq 0
\end{aligned}
$$

```
Model = ConcreteModel()

Model.x = Var([1,2], domain=NonNegativeReals)

Model.obj = Objective(expr = 3*Model.x[1] + 2*Model.x[2], sense=maximize)

Model.res1 = Constraint(expr = 2*Model.x[1] + Model.x[2] <= 100)

Model.res2 = Constraint(expr = Model.x[1] + Model.x[2] <= 80)

Model.res3 = Constraint(expr = Model.x[1] <= 40)

SolverFactory('glpk').solve(Model)

Model.display()
```

  - **Nota**: si en el campo donde aparece *“sense=maximize”* no se especifica nada, por defecto Pyomo assume que estamos minimizando.

Ejemplos

- Caso Proyectos:



$$x_i = \begin{cases}0\\1\end{cases}$$

$$max \boxed{\sum_i} g_i * x_i$$

$$\boxed{\sum_i} x_i = 2$$

```

from __future__ import division
from pyomo.environ import *

from pyomo.opt import SolverFactory

Model = ConcreteModel()

# Sets and Parameters
numProyectos=8

#p=[1, 2, 3, 4, 5, 6, 7, 8]
p=RangeSet(1, numProyectos)

valor={1:2, 2:5, 3:4, 4:2, 5:6, 6:3, 7:1, 8:4}

# Variables
Model.x = Var(p, domain=Binary)

# Objective Function
Model.obj = Objective(expr = sum(Model.x[i]*valor[i] for i in p), sense=maximize)

# Constraints
Model.res1 = Constraint(expr = sum(Model.x[i] for i in p) == 2)

# Applying the solver
SolverFactory('glpk').solve(Model)

Model.display()

```

Ejemplos

- Caso Mínimo Costo:

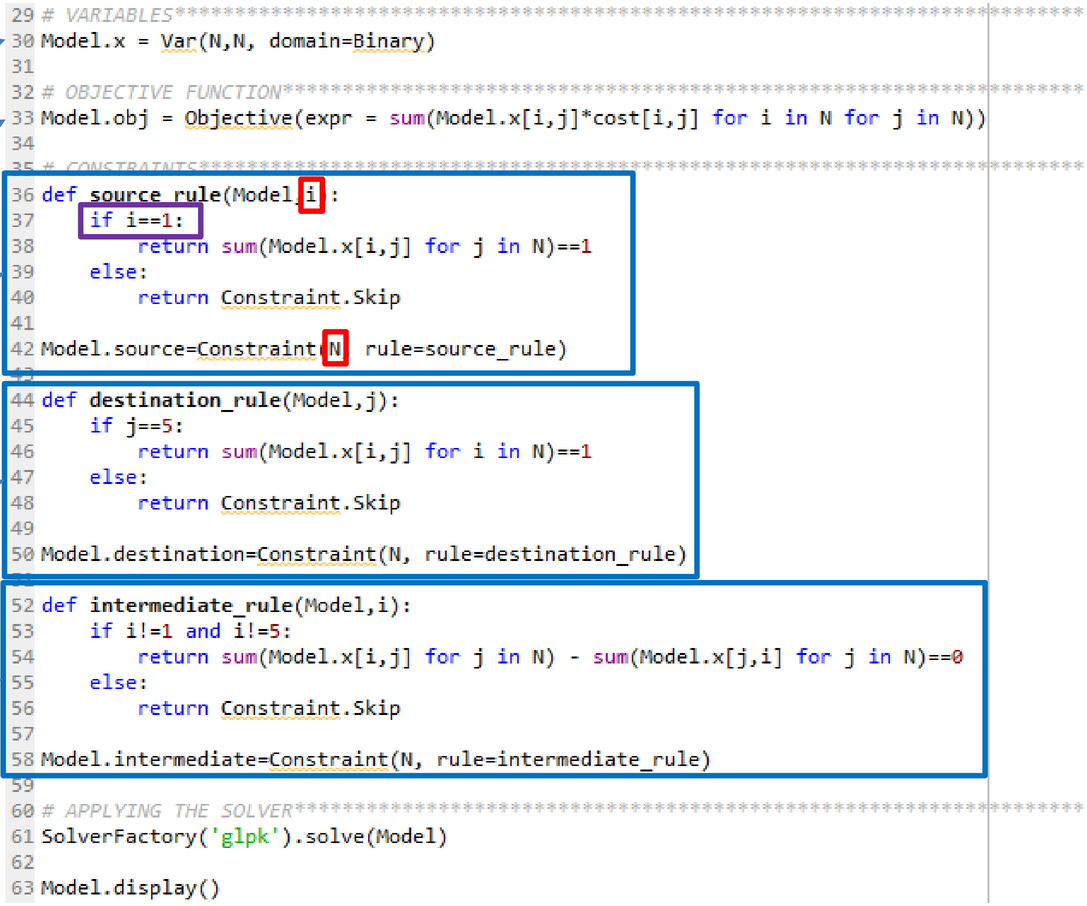
$$X_{ij} = \begin{cases} 1 \rightarrow \text{Escojo el enlace } (i,j) \\ 0 \rightarrow \text{No Escojo el enlace } (i,j) \end{cases}$$

$$\min \sum_{i \in N} \sum_{j \in N} C_{ij} X_{ij}$$

$$\sum_{j \in N} X_{ij} = 1 \quad \boxed{\forall i}\boxed{/ i = 1}$$

$$\sum_{i \in N} X_{ij} = 1 \quad \forall j / j = 5$$

$$\sum_{j \in N} X_{ij} - \sum_{j \in N} X_{ji} = 0 \quad \forall i / i \notin \{1,5\}$$

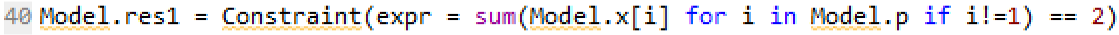


```
# VARIABLES*************************************************************************
Model.x = Var(N,N, domain=Binary)

# OBJECTIVE FUNCTION*****************************************************************
Model.obj = Objective(expr = sum(Model.x[i,j]*cost[i,j] for i in N for j in N))

# CONSTRAINTS************************************************************************
def source_rule(Model,i):
    if i==1:
        return sum(Model.x[i,j] for j in N)==1
    else:
        return Constraint.Skip

Model.source=Constraint(N, rule=source_rule)

def destination_rule(Model,j):
    if j==5:
        return sum(Model.x[i,j] for i in N)==1
    else:
        return Constraint.Skip

Model.destination=Constraint(N, rule=destination_rule)

def intermediate_rule(Model,i):
    if i!=1 and i!=5:
        return sum(Model.x[i,j] for j in N) - sum(Model.x[j,i] for j in N)==0
    else:
        return Constraint.Skip

Model.intermediate=Constraint(N, rule=intermediate_rule)

# APPLYING THE SOLVER****************************************************************
SolverFactory('glpk').solve(Model)

Model.display()
```

Tips

- Cómo introducimos un *“tal que”* en una sumatoria?
  - Forma 1:

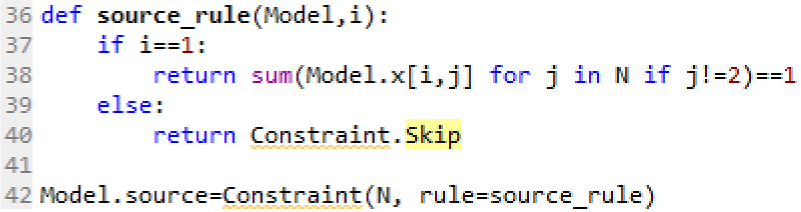
$$\sum_{i/i\neq 1}^{p} x_i = 2$$



```
Model.res1 = Constraint(expr = sum(Model.x[i] for i in Model.p if i!=1) == 2)
```

  - Forma 2:

$$\sum_{j\in N/j\neq 2} x_{ij} = 1 \quad \forall i \in N|i = 1$$

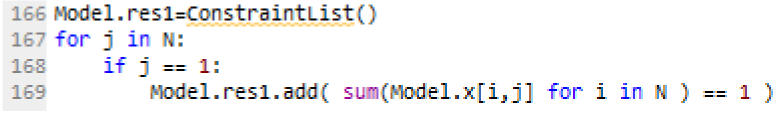


```
def source_rule(Model,i):
    if i==1:
        return sum(Model.x[i,j] for j in N if j!=2)==1
    else:
        return Constraint.Skip

Model.source=Constraint(N, rule=source_rule)
```

Tips

- Otro método para crear una restricción que tenga un ‘para todo’:
  - Ejemplo 1:

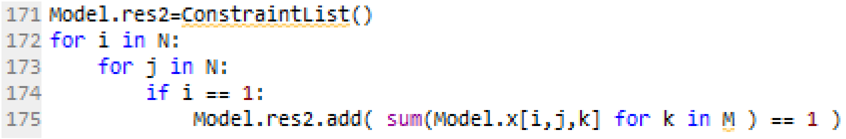
$$\sum_{\mathrm{i} \in N} x_{ij} = 1 \quad \forall \mathrm{j} \in N | \mathrm{j} = 1$$



```
Model.res1=ConstraintList()
for j in N:
    if j == 1:
        Model.res1.add( sum(Model.x[i,j] for i in N ) == 1 )
```

  - Ejemplo 2:

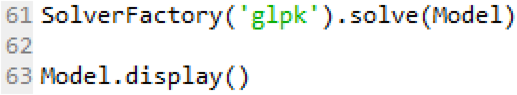
$$\sum_{\mathrm{k} \in M} x_{ijk} = 1 \quad \forall \mathrm{i}, \mathrm{j} \in N | i = 1$$



```
Model.res2=ConstraintList()
for i in N:
    for j in N:
        if i == 1:
            Model.res2.add( sum(Model.x[i,j,k] for k in M ) == 1 )
```

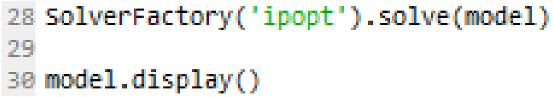
Solvers

- Problemas LP y MIP:



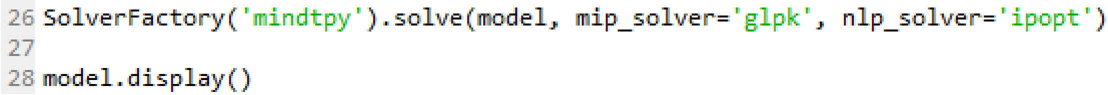
```
SolverFactory('glpk').solve(Model)

Model.display()
```

- Problemas NLP:



```
SolverFactory('ipopt').solve(model)

model.display()
```

- Problemas MINLP:



```
SolverFactory('mindtpy').solve(model, mip_solver='glpk', nlp_solver='ipopt')

model.display()
```