```
!pip install -q amplpy
from amplpy import AMPL, tools
ampl = tools.ampl_notebook(
    modules=["coin"],
    license_uuid="default")
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.4/2.4 MB 20.0 MB/s eta 0:00:00
Using default Community Edition License for Colab. Get yours at: https://ampl.com/ce
Licensed to AMPL Community Edition License for the AMPL Model Colaboratory (https://ampl.com/colab).
```

```
# Maximum Flow
%%writefile MaxFlow.mod
param m;

set I={1..m};
set J = {1..m};
param capacity{I,J};

var x{I,J} >=0;

maximize Z: x[m,1];

s.t. Tranship {j in J}: sum{i in I} x[i,j] - sum{i in I} x[j,i] =0;

s.t. Capacity{i in I, j in J}: x[i,j] <= capacity[i,j]; # the flow in the arc (i,j) should not exceed the capacity of the arc
```

```
Writing MaxFlow.mod
```

```
%%writefile MaxFlow.dat

param m := 6;

param capacity :  1    2    3    4    5    6 :=
1               0   500  400    0    0    0
2               0     0    0  300  250    0
3               0     0    0  200  150    0
4               0     0    0    0    0  400
5               0     0    0    0    0  350
6            9999    0    0    0    0    0 ;
```

```
Writing MaxFlow.dat
```

```
%%ampl_eval

reset;

# Model File
model MaxFlow.mod;
data MaxFlow.dat;

# Calling Optimization Engine and Optimizing
option solver cbc;

solve;

# Display Results
display Z,x;
```

```
cbc 2.10.12: optimal solution; objective 750
0 simplex iterations
Z = 750

x [*,*]
:    1     2     3     4     5     6     :=
1     0   400   350     0     0     0
2     0     0     0   200   200     0
3     0     0     0   200   150     0
4     0     0     0     0     0   400
5     0     0     0     0     0   350
6   750     0     0     0     0     0
;
```

```
%%writefile Transportation.mod

set S = {1,2,3}; #source nodes
set D = {"A", "B", "C", "D"}; # destination nodes

param C{S,D}; # cost matrix
param Supply{S}; # Supply capacity of the supply nodes
param Demand{D}; # Demand at nodes in set D

var x{S,D} >=0; # transportation quantity from node set S to demand set D

minimize Z: sum{i in S, j in D} C[i,j]*x[i,j]; # Total Transportation
s.t. supply{i in S}: sum{j in D} x[i,j] <= Supply[i]; # supply constraints
s.t. demand{j in D}: sum{i in S} x[i,j] = Demand[j]; # Demand constraints
```

```
Overwriting Transportation.mod
```

```
%%writefile Transportation.dat

param C :   A   B   C   D :=
1          7  11   3   2
2          1   6   0   1
3          9  15   8   5 ;

param Supply :=
1  6
2  1
3 10 ;

param Demand :=
A 2
B 3
C 5
D 6 ;
```

```
Overwriting Transportation.dat
```

```
%%ampl_eval

reset;

# Model File
model Transportation.mod;
data Transportation.dat;

# Calling Optimization Engine and Optimizing
option solver cbc;

solve;

# Display Results
display Z,x;
```

```
cbc 2.10.12: optimal solution; objective 95
0 simplex iterations
Z = 95

x :=
1 A   0
1 B   1
1 C   5
1 D   0
2 A   0
2 B   1
2 C   0
2 D   0
3 A   2
3 B   1
3 C   0
3 D   6
;
```

```
# Assignment Problem

%%writefile Assignment.mod

set I;   # outbound flights (CLT -> MCO)
set J;   # return flights (MCO -> CLT)

param C{I,J};   # waiting-time cost (use 99 for infeasible)

var x{I,J} binary;   # 1 if outbound i is matched to return j

minimize Z:
    sum{i in I, j in J} C[i,j] * x[i,j];

s.t. assign_out{i in I}:
    sum{j in J} x[i,j] = 1;

s.t. assign_back{j in J}:
    sum{i in I} x[i,j] = 1;
```

```
Overwriting Assignment.mod
```

```
%%writefile Assignment.dat

set I := 1 2 3 4 5 6 7;
set J := 1 2 3 4 5 6 7;


param C :  1   2   3   4   5   6   7 :=
1        99  99  99   1   2   4   5
2         1  99  99  99  99   3  99
3         2  99  99  99  99   2  99
4        99   2  99  99  99  99   1
5         6   4   2  99  99  99  99
6        99  99   4   3   2  99  99
7        10   8  99   5   4   2   1 ;
```

```
Overwriting Assignment.dat
```

```
%%ampl_eval

reset;

# Model File
model Assignment.mod;
data Assignment.dat;

# Calling Optimization Engine and Optimizing
option solver cbc;

solve;

# Display Results
display Z,x;
```

```
cbc 2.10.12: optimal solution; objective 11
0 simplex iterations
Z = 11

x [*,*]
:   1   2   3   4   5   6   7    :=
1   0   0   0   1   0   0   0
2   1   0   0   0   0   0   0
3   0   0   0   0   0   1   0
4   0   1   0   0   0   0   0
5   0   0   1   0   0   0   0
6   0   0   0   0   1   0   0
7   0   0   0   0   0   0   1
;
```