Knapsack Problem Julia/JuMP Walk-Through

Julia Version 0.4.6, JuMP Version 0.13.2

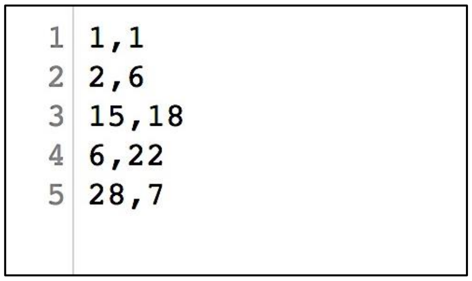
Problem Setup and Formulation

We have 5 objects and a knapsack. Each object i has positive weight and positive value. The knapsack has a weight capacity. We wish to select a set of objects to put in the knapsack so that we maximize the total value of the objects that we carry in the knapsack.

Consider that the capacity of the knapsack is 11.

Weight and value information for each item is shown in the table below. This weight and value information is also stored in a CSV file called **knapsack_data.csv**. This file is shown below the table.

Weights	Values
1	1
2	6
15	18
6	22
28	7



```
1,1
2,6
15,18
6,22
28,7
```

knapsack_data.csv

Problem Formulation:

Decision Variables

x_1	= 1 if we select item 1 for our knapsack, 0 otherwise
x_2	= 1 if we select item 2 for our knapsack, 0 otherwise
x_3	= 1 if we select item 3 for our knapsack, 0 otherwise
x_4	= 1 if we select item 4 for our knapsack, 0 otherwise
x_5	= 1 if we select item 5 for our knapsack, 0 otherwise

Constraints

$x_1 + 2x_2 + 15x_3 + 6x_4 + 28x_5 \leq 11$ The weight of our knapsack cannot exceed 11

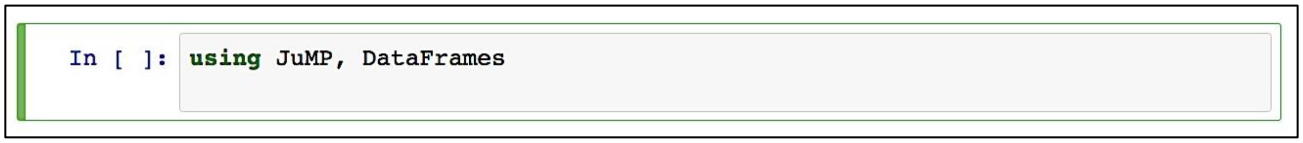
Objective Function

$Max \quad x_1 + 6x_2 + 18x_3 + 22x_4 + 7x_5$ Maximize the total value of our knapsack.

Julia/JuMP Walk-Through

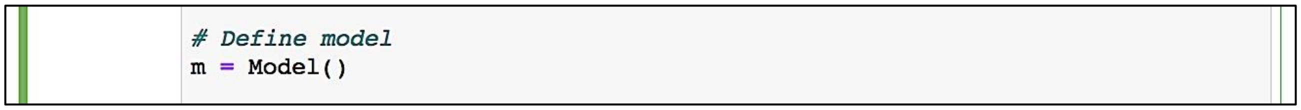
Model Setup

Begin by ***specifying the packages and/or solver*** that you will be using in your Julia program. In this class, begin by calling the packages **`JuMP`** and **`DataFrames`** (since we will be reading data from a CSV file)



```
In [ ]: using JuMP, DataFrames
```

Next, ***define and name your model***. For our Knapsack Problem example, we will define a base model named **m** that uses the default solver.

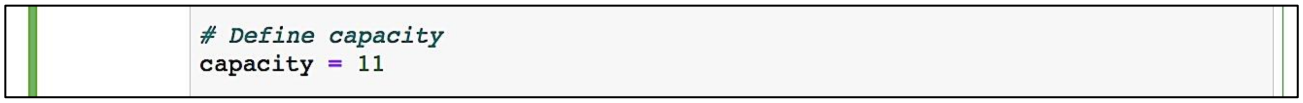


```
# Define model
m = Model()
```

Data Setup

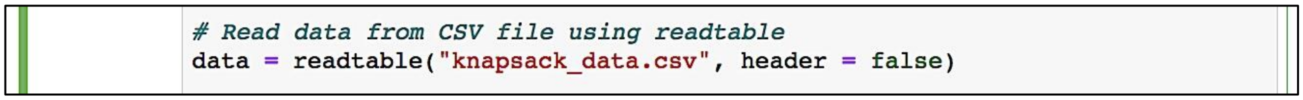
For many optimization problems, you will be given starting information or data. Before you can define decision variables and constraints, you will need to ***create the data*** in Julia. Data can be stored in arrays (lists), matrices, tuples, or dictionaries, etc.

In our Knapsack Problem example, let us first define the capacity of our knapsack.



```
# Define capacity
capacity = 11
```

Next, we are given starting data about the weights and values of the five objects that we can choose from in the form of a CSV file. To import data from the CSV file, we will use the **`readtable`** function to create a table named **data**.



```
# Read data from CSV file using readtable
data = readtable("knapsack_data.csv", header = false)
```

From the CSV file, we see that the weights of the objects are the *first column of the table*, and their values are the *second*. By indexing into the table named **data**, we will create two *arrays* containing the weights and the values of the five objects. Remember that indexing goes as **`[row,column]`**. For the weights, we want all rows and the first column of the table. For the values, we want all rows and the second column of the table.

```
# Weights from first column, weights = [1 2 15 6 28]
weights = data[:,1]

# Values from second column, values = [1 6 18 22 7]
values = data[:,2]
```

Note that we could have also used the function **`readcsv`** when reading the CSV file. This would have made the object **data** into a multidimensional array instead of a table. This, however, would not have affected the way we indexed into **data** to obtain arrays for the weights and values.

```
# Read data from CSV file using readcsv
data = readcsv("knapsack_data.csv", header = false)
```

Decision Variables

After setting up the starting data for our LP, we can now ***define our decision variables*** using the **`@variable`** macro. You must specify the model, the decision variable name and indices (if applicable), any upper and lower bounds, and other variable classifications (binary/integer).

Below we define five decision variables. The decision variables are named **x** with one index. We specify the indices by writing the name of the decision variable followed by brackets containing the range of integers 1 to 5, as denoted by **`1:5`**. Since an item is either in or not in our knapsack, the decision variables will be binary.

Decision Variables: x_1 x_2 x_3 x_4 x_5

```
# Assign binary values to items
@variable(m, x[1:5], Bin)
```

Constraints

To ***add constraints***, use the **`@constraint`** macro. You must specify the model name, followed by the constraint expression. Below we add the weight constraint for our knapsack. Note how we use indexing to obtain the proper weight coefficient and decision variable in our constraint expression.

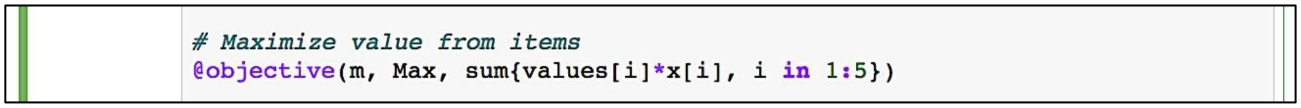
$x_1 + 2x_2 + 15x_3 + 6x_4 + 28x_5 \leq 11$ Weight cannot exceed capacity of 11.

```
# Constraint on total weight
@constraint(m, sum{weights[i]*x[i], i in 1:5} <= capacity)
```

Objective Function

To add an ***objective function***, use the **`@objective`** macro. You must specify the model name, whether it is **`Min`** or **`Max`**, and the objective function expression. Below we add the objective that we want to maximize the total value of our knapsack.

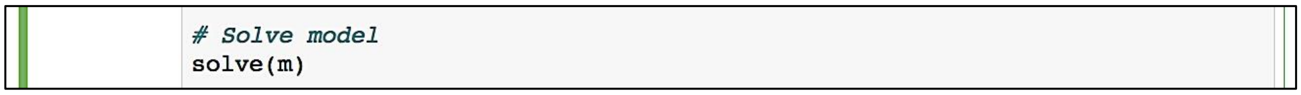
$$Max \qquad x_1 + 6x_2 + 18x_3 + 22x_4 + 7x_5$$



```
# Maximize value from items
@objective(m, Max, sum{values[i]*x[i], i in 1:5})
```

Solving the Model and Model Output

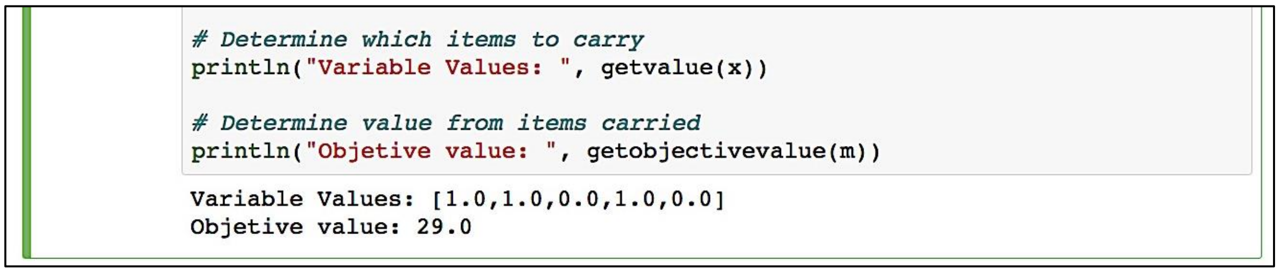
To ***solve the model***, use the **`solve()`** function and write the name of the model you wish to solve within the parentheses.



```
# Solve model
solve(m)
```

There are various ways to ***print the model's output***. Below are a few examples of how to print model output.

To print all decision variable values and objective value. Here we use the functions **`getvalue()`** and **`getobjectivevalue()`** where in the parentheses we write the decision variable and model respectively.

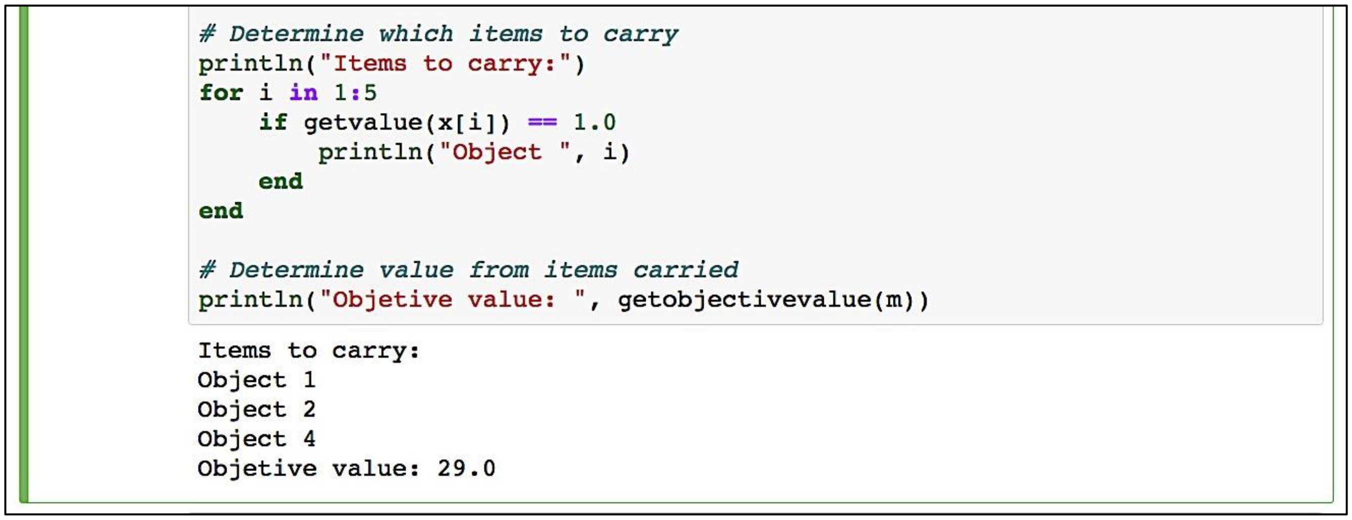


```
# Determine which items to carry
println("Variable Values: ", getvalue(x))

# Determine value from items carried
println("Objetive value: ", getobjectivevalue(m))
```

```
Variable Values: [1.0,1.0,0.0,1.0,0.0]
Objetive value: 29.0
```

To print only the decision variable values that are non-zero and the final objective value. Here we use a **`for`** loop and a nested **`if`** statement within the for loop.

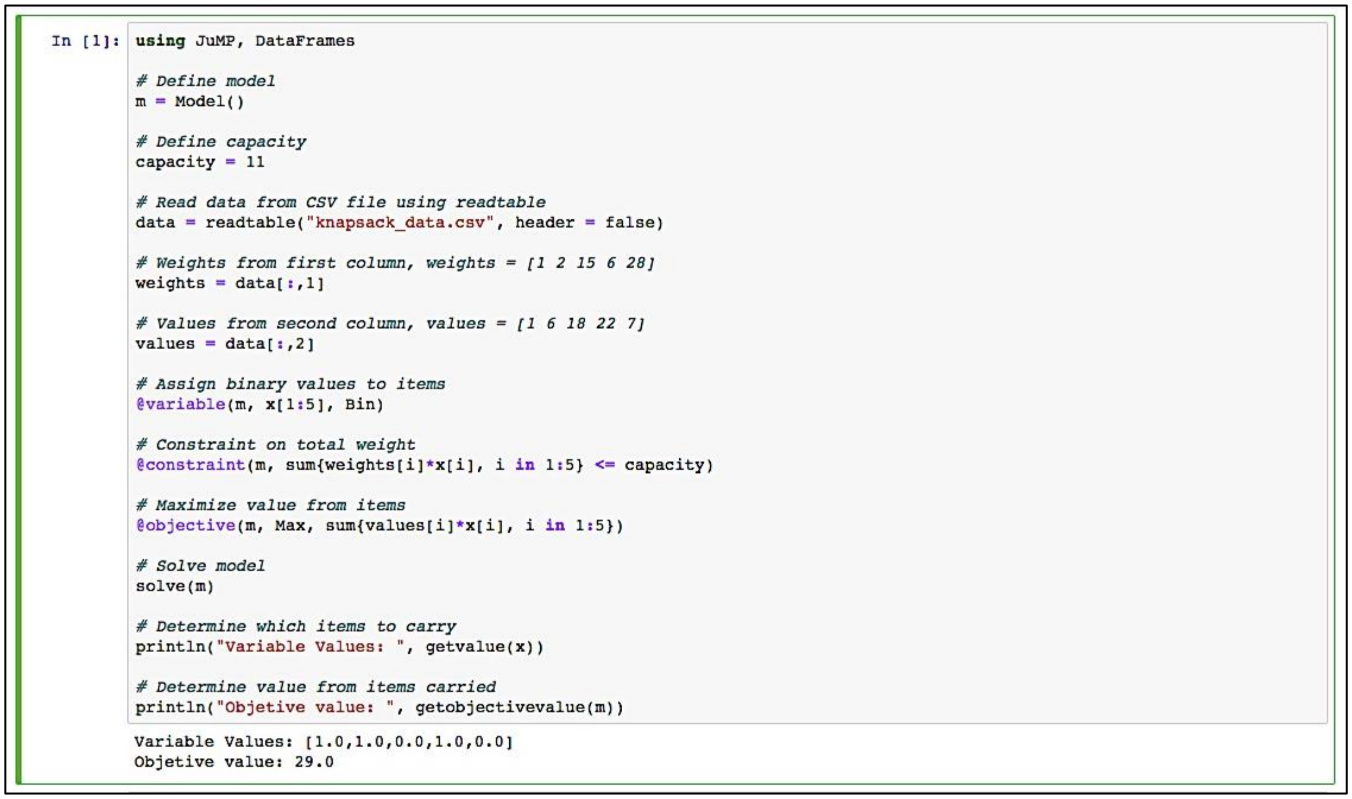


```
# Determine which items to carry
println("Items to carry:")
for i in 1:5
    if getvalue(x[i]) == 1.0
        println("Object ", i)
    end
end

# Determine value from items carried
println("Objetive value: ", getobjectivevalue(m))
```

```
Items to carry:
Object 1
Object 2
Object 4
Objetive value: 29.0
```

<u>Full Julia Code and Output for the Diet Problem</u>



```
In [1]: using JuMP, DataFrames

        # Define model
        m = Model()

        # Define capacity
        capacity = 11

        # Read data from CSV file using readtable
        data = readtable("knapsack_data.csv", header = false)

        # Weights from first column, weights = [1 2 15 6 28]
        weights = data[:,1]

        # Values from second column, values = [1 6 18 22 7]
        values = data[:,2]

        # Assign binary values to items
        @variable(m, x[1:5], Bin)

        # Constraint on total weight
        @constraint(m, sum{weights[i]*x[i], i in 1:5} <= capacity)

        # Maximize value from items
        @objective(m, Max, sum{values[i]*x[i], i in 1:5})

        # Solve model
        solve(m)

        # Determine which items to carry
        println("Variable Values: ", getvalue(x))

        # Determine value from items carried
        println("Objetive value: ", getobjectivevalue(m))
```

```
Variable Values: [1.0,1.0,0.0,1.0,0.0]
Objetive value: 29.0
```