Micro-Breweri

This is the first assignment that involves **storage** and utilization of a balance constraint. It is important to understand this assignment since you will see some more complicated versions of it later.

Problem

- Minimize the storage cost while satisfying the demand and planning the production

Sets

- Months: $m \in Months = \{jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec\}$.

Parameters

- $Demand_m$: The number of liters of beer you have to deliver to the cafe's
- $BrewCap$: No. of liters of beer you at most can brew pr. month
- $StoreCap$: No. of liters of beer you at most can store in the basement
- $Cost$: Cost pr. liter for storing beers in the basement (to your mum)

Decision variables

- Amount of beer produced each month m: x_m.
- Amount of beer stored each month m: y_m.

Model

Objective:

- Total storage costs to be minimized:

$$\sum_m Cost \cdot y_m$$

Constraints:

- Brew capacity limitation:

$$x_m \leq BrewCap \; \forall \; m$$

- Storage capacity limitation:

$$y_m \leq StoreCap \; \forall \; m$$

- Storage (balance) constraint, what goes in, must come out:

$$y_{m-1}|(m > 1) + x_m - Demand_m = y_m \; \forall \; m$$

  The first part: $y_{m-1}|(m > 1)$, inserts the storage from the previous month, **if** it is not the first month. Further explanation below.

The above model has one objective function and three abstract constraints. The two first can however just as well be implemented by setting the variables x_m and y_m. Furthermore, notice that the **balance** constraint, the last constraint, links the variables and the demand, such that a separate demand constraint is not necessary. Notice also, that the storage in the **end** of month m (i.e. y_m) is equal to what was on storage at the end of the previous month (i.e. y_{m-1}) plus what is produced in this month (i.e. x_m) minus what we delivered this month (i.e. the $Demand_m$). It is very important that you understand this constraint because it will show up in more complex forms in the coming assignments.

The full model in Julia/JuMP, available with the name

```
Microbrewery.jl
```

from the book web-site, is given below:

```julia
#************************************************************************
# MicroBreweri Assignment, "Mathematical Programming Modelling" (42112)
using JuMP
using HiGHS
#************************************************************************

#************************************************************************
# PARAMETERS
```

```
Months = ["Jan","Feb","Mar","Apr","May",
          "Jun","Jul","Aug","Sep","Oct","Nov","Dec"]

# prod and store data
ProdCap = 120

StoreCap = 200
StoreCostPrLiter = 1

# demand from cafes of beers
Demand = [15 30 25 55 75 115 190 210 105 65 20 20]

# size of sets
M = length(Months)
#************************************************************************

#************************************************************************
# Model
microbreweri = Model(HiGHS.Optimizer)

@variable(microbreweri, 0 <= x[1:M] <= ProdCap) # production
@variable(microbreweri, 0 <= y[1:M] <= StoreCap) # storage in the end of the month

# Minimize storage cost
@objective(microbreweri, Min,
           sum( StoreCostPrLiter*y[m] for m=1:M))

# storage balance constraint
@constraint(microbreweri, [m=1:M],
            y[m] == (m>1 ? y[m-1] : 0) + x[m] - Demand[m])
print(microbreweri)
#************************************************************************

#************************************************************************
# solve
optimize!(microbreweri)
println("Termination status: $(termination_status(microbreweri))")
#************************************************************************

#************************************************************************
# Report results
println("-------------------------------------");
if termination_status(microbreweri) == MOI.OPTIMAL
  println("RESULTS:")
  println("Objective: $(objective_value(microbreweri))")
  for m = 1:length(Months)
```

```
    println(" production $(value(x[m]))  store     : $(value(y[m]))")
  end
else
  println("  No solution")
end
println("--------------------------------------");
#**************************************************************************
```

Comments to the Julia/JuMP program:

- Notice: The use of **conditional parts of a constraint**:

  ```
  (m>1 ? y[m-1] : 0)
  ```

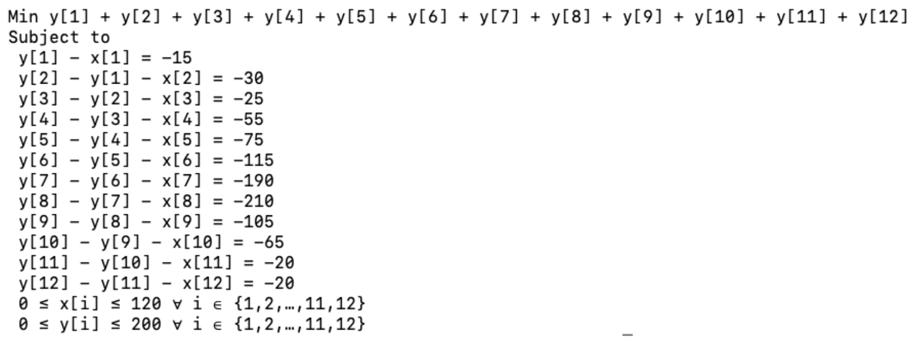
- Notice: The condition means that the constraint for the first month becomes: (0 because there is no initial storage)

  ```
  y[m] == 0 + x[m] - Demand[m])
  ```

- Notice: The condition means that the constraint for the remaining month becomes:

  ```
  y[m] == y[m-1] + x[m] - Demand[m])
  ```

The effect of the conditional part is also clear when printing the model, which can be seen in the first constraint:



```
Min y[1] + y[2] + y[3] + y[4] + y[5] + y[6] + y[7] + y[8] + y[9] + y[10] + y[11] + y[12]
Subject to
 y[1] - x[1] = -15
 y[2] - y[1] - x[2] = -30
 y[3] - y[2] - x[3] = -25
 y[4] - y[3] - x[4] = -55
 y[5] - y[4] - x[5] = -75
 y[6] - y[5] - x[6] = -115
 y[7] - y[6] - x[7] = -190
 y[8] - y[7] - x[8] = -210
 y[9] - y[8] - x[9] = -105
 y[10] - y[9] - x[10] = -65
 y[11] - y[10] - x[11] = -20
 y[12] - y[11] - x[12] = -20
 0 ≤ x[i] ≤ 120 ∀ i ∈ {1,2,…,11,12}
 0 ≤ y[i] ≤ 200 ∀ i ∈ {1,2,…,11,12}
```