Homework 0: Exercises for Julia Tutorial

After you have gone through the tutorial, you should be able to fill in these simple exercises. To complete the homework, please take the following steps:

1. Install Julia in your local computing environment
2. Decide on your development environment (IJulia, VSCode, or another)
3. Be sure to run all of the cells
4. Print a pdf of the notebook, ensuring that everything rendered correctly
5. Upload the pdf to Gradescope, and match the pages of the PDF to the problems in the assignment.

Exercises

Get started by reading the following statements about your class work and signing that you understand and agree by typing your name at the bottom of this cell.

1. You are encouraged to discuss homework problems with classmates and even work in groups.

2. However, **the work you turn in must be your own**. You must not communicate files containing code or answers to homework questions to each other.

3. Many homeworks require the use of Julia and JuMP. We'll provide instructions to help you install this programming environment on your own computer, together with tutorials and exercises, but ultimately it is your responsibility to ensure that you have a stable platform on which to develop and run Julia code and jupyter notebooks.

4. Submission of all homeworks will be through Gradescope. You will be required to submit a **pdf printout of your jupyter notebook**, NOT source code.

   a. Please submit the answers **in the same order as on the assignment**

   b. It is useful to denote the start of each question in your notebook using a large font, for example:

   # Question 1a

   this will help you do the matching in the Gradescope submission tool

5. You can learn to do optimization modeling only by doing it yourself, not by following along what others are doing!

6. **PLAGIARISM AND OTHER TYPES OF ACADEMIC MISCONDUCT IS NOT TOLERATED AND WILL HAVE CONSEQUENCES.** Please read this information about UW's definition of academic misconduct.

1.0 Signature

PLEASE ACKNOWLEDGE YOUR UNDERSTANDING AND ACCEPTANCE OF THE RULES OF THE COURSE BY TYPING YOUR NAME IN THIS MARKDOWN CELL HERE: Damion Huppert

1.1 Working with arrays and tuples

Define the array

```
square = [1, 2, 3]
```

and the tuple

```
round = (4,5,6)
```

In [1]:
```
square = [1, 2, 3]
round = (4, 5, 6)
```

Out[1]:
```
(4, 5, 6)
```

Access the first element of the array and the tuple, and add them together.

In [5]:
```
square[1] + round[1]
```

Out[5]:
```
5
```

Change the first element of the array to be equal to the first element of the tuple.

In [6]:
```
square[1] = round[1]
```

Out[6]:
```
4
```

Try to change the third element of the tuple to be equal to that of the array.

Why will this not work?
Tuples are immutable

In [ ]:
```
# This will not work, because ...
#  tuples are immutable
```

1.2 Dictionaries

Create a dictionary which lists three of your favorite restaurants and their ranking (1, 2 or 3).

In [8]:
```
restaurant_rankings = Dict(1 => "Senjoy", 2 => "Mooyah", 3 => "Chipotle")
```

Out[8]:
```
Dict{Int64, String} with 3 entries:
  2 => "Mooyah"
  3 => "Chipotle"
  1 => "Senjoy"
```

1.3 Matrices (two-dimension arrays)

Create the following Julia matrix:

$$B = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 0 & 1 \\ 0 & 2 & 4 \end{bmatrix}$$

(Please also look in this markdown cell about how to write matrices in your notebooks)

In [12]:
```
B = [1 2 1; 3 0 1; 0 2 4]
```

Out[12]:
```
3×3 Matrix{Int64}:
 1  2  1
 3  0  1
 0  2  4
```

Change the first element in the first row of B into 5, and check if it is an even number. (Tip: Use "a%b == c", which means the remainder of a/b is c)

In [14]:
```
B[1, 1] = 5
B[1, 1] % 2 == 0
```

Out[14]:
```
false
```

1.4 For loops

Write a for loop to print the integers from 1 to 5.

In [17]:
```
for i in 1:5
    println(i)
end
```

```
1
2
3
4
5
```

Now write a for loop to go through every element in the above matrix B, check if it is odd. If it is, then add 1 to that element of the matrix, and print your resulting matrix.

In [23]:
```
result = []
for i in B
    if (i%2==1)
        push!(result, i)
    end
end
```

```
println(result)
```

```
Any[5, 3, 1, 1]
```

1.5 List Comprehensions

Create a list (vector) of integers called my_list that contains the integers 1 to 10 in one line using a list comprehension

In [25]:
```
my_list = [i for i in 1:10]
```

Out[25]:
```
10-element Vector{Int64}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
```

Create a list (vector) called my_list2 of the square of all odd numbers from 13 to 17 using a list comprehension

In [26]:
```
my_list2 = [i^2 for i in 13:17 if i%2 == 1]
```

Out[26]:
```
3-element Vector{Int64}:
 169
 225
 289
```

1.6 Functions

Write a function called `my_func` which takes a number as an input, and return an array containing integers from 1 to n. And try your function with input 5.

In [29]:
```
my_func(x) = [i for i in 1:x]
my_func(5)
```

Out[29]:
```
5-element Vector{Int64}:
 1
 2
 3
 4
 5
```

1.7 Plots

This problem is designed to ensure that you can plot using Julia and print the plots

install the "Plots" package in your Julia environment by executing the cell below

In [30]:
```
using Pkg
Pkg.add("Plots")
```

```
  Resolving package versions...
 No Changes to `C:\Users\damio\.julia\environments\v1.11\Project.toml`
 No Changes to `C:\Users\damio\.julia\environments\v1.11\Manifest.toml`
```

Check the status of your packages by running the command in the cell below

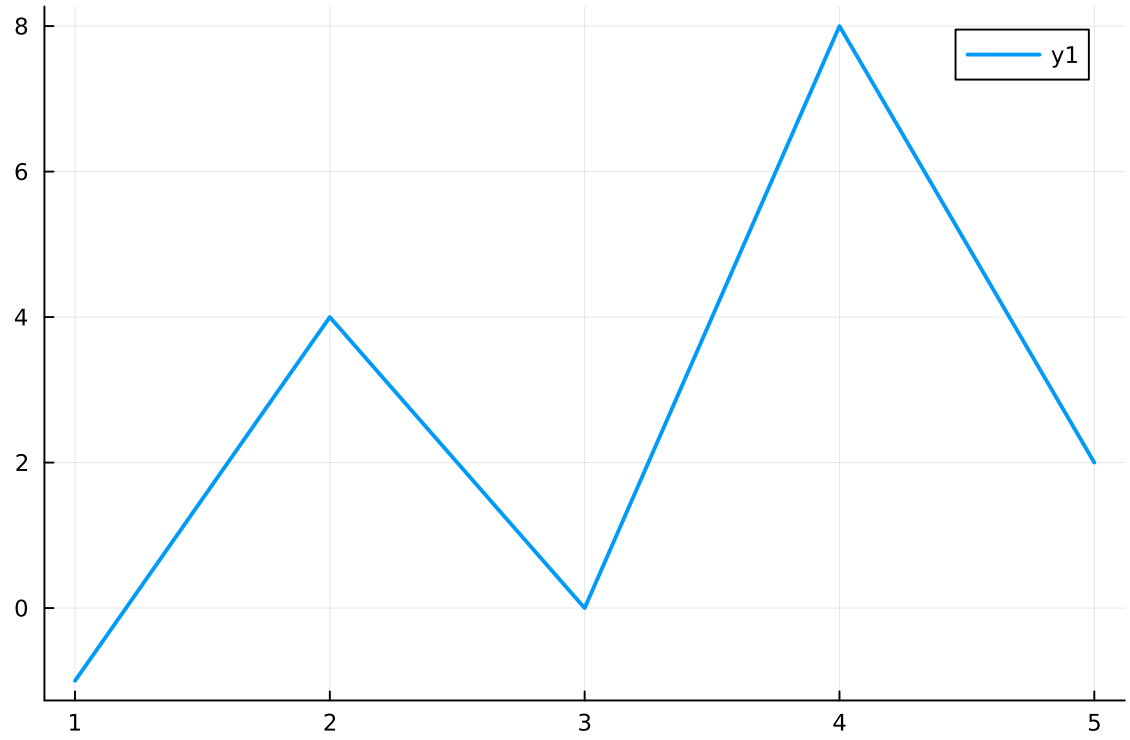
In [31]:
```
Pkg.status()
```

```
Status `C:\Users\damio\.julia\environments\v1.11\Project.toml`
  [7073ff75] IJulia v1.26.0
  [4076af6c] JuMP v1.23.6
  [91a5bcdd] Plots v1.40.9
```

Now make a plot by executing the cell below. This will make a simple linegraph using the Julia Plots package

In [34]:
```
using Plots
x = 1 : 5
y = [-1, 4, 0, 8, 2]
plot(x,y,lw=2)
```

Out[34]:



1.8 LaTeX

This is to ensure that you can write and print mathematics in your Markdown code as part of your submissions.

This is an example of markdown. We can make a list like this

- Item 1
- Item 2
- Item 3

And LaTeX by putting a single dollar sign $x_1 = 4$, to put it inline

And double dollar signs to make it its own line

$$\sum_{j=1}^{8} \sin(x) \leq 14$$

You can even do some alignment of equations:

$$x_1 = 4 \tag{1}$$
$$x_2 = 18 \tag{2}$$
$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad \forall i \in M \tag{3}$$

And do matrices:

$$A = \begin{bmatrix} 1 & 7 & 3 \\ -1 & \alpha & \beta \end{bmatrix}$$

Look at the markdown in this cell to ensure you know how to do this

In [ ]:

In [ ]:

An introduction to JuMP

Welcome! This tutorial will introduce you to the basics of the latest stable version of **JuMP**. If you haven't yet, work through the tutorial on Julia and do the exercises first.

WARNING: This tutorial covers material that will be discussed in class in the first couple of weeks. If something feels confusing, don't worry too much right now.

Some useful resources as you are learning JuMP are:

- JuMP documentation
- Textbook: Julia Programming for Operations Research
- the Discourse forum

2.0: Jump setup

Before we start, let's be sure we have installed the following Julia packages (we'll explain

more about what those packages are later!).

In [35]:
```
using Pkg
Pkg.add("Ipopt")
Pkg.add("HiGHS")
Pkg.add("Interact")
Pkg.add("Plots")
```

```
   Resolving package versions...
   Installed Hwloc_jll ──────── v2.11.2+3
   Installed SPRAL_jll ──────── v2024.5.8+0
   Installed OpenBLAS32_jll ─── v0.3.29+0
   Installed MUMPS_seq_jll ──── v500.700.301+0
   Installed ASL_jll ────────── v0.1.3+0
   Installed Ipopt_jll ──────── v300.1400.1700+0
   Installed METIS_jll ──────── v5.1.3+0
   Installed Ipopt ──────────── v1.7.1
    Updating `C:\Users\damio\.julia\environments\v1.11\Project.toml`
  [b6b21f68] + Ipopt v1.7.1
    Updating `C:\Users\damio\.julia\environments\v1.11\Manifest.toml`
  [b6b21f68] + Ipopt v1.7.1
  [ae81ac8f] + ASL_jll v0.1.3+0
  [e33a78d0] + Hwloc_jll v2.11.2+3
  [9cc047cb] + Ipopt_jll v300.1400.1700+0
  [d00139f3] + METIS_jll v5.1.3+0
  [d7ed1dd3] + MUMPS_seq_jll v500.700.301+0
  [656ef2d0] + OpenBLAS32_jll v0.3.29+0
⌅ [319450e9] + SPRAL_jll v2024.5.8+0
        Info Packages marked with ⌅ have new versions available but compatibility co
nstraints restrict them from upgrading. To see why use `status --outdated -m`
Precompiling project...
   2392.9 ms  ✓ ASL_jll
   2380.5 ms  ✓ OpenBLAS32_jll
   2391.3 ms  ✓ Hwloc_jll
   2537.5 ms  ✓ METIS_jll
   1819.1 ms  ✓ MUMPS_seq_jll
   2280.7 ms  ✓ SPRAL_jll
   4947.3 ms  ✓ Ipopt_jll
   7639.9 ms  ✓ Ipopt
  8 dependencies successfully precompiled in 18 seconds. 211 already precompiled.
   Resolving package versions...
   Installed HiGHS ──────── v1.13.0
   Installed HiGHS_jll ──── v1.9.0+0
    Updating `C:\Users\damio\.julia\environments\v1.11\Project.toml`
  [87dc4568] + HiGHS v1.13.0
    Updating `C:\Users\damio\.julia\environments\v1.11\Manifest.toml`
  [87dc4568] + HiGHS v1.13.0
  [8fd58aa0] + HiGHS_jll v1.9.0+0
Precompiling project...
    431.5 ms  ✓ HiGHS_jll
   6936.0 ms  ✓ HiGHS
  2 dependencies successfully precompiled in 8 seconds. 219 already precompiled.
   Resolving package versions...
   Installed ColorTypes ─────────── v0.11.5
   Installed XZ_jll ─────────────── v5.6.4+1
   Installed ColorVectorSpace ───── v0.10.0
   Installed Pidfile ────────────── v1.3.0
   Installed WebSockets ─────────── v1.6.0
   Installed AssetRegistry ──────── v0.1.0
   Installed Colors ─────────────── v0.12.11
   Installed Knockout ───────────── v0.2.6
   Installed Widgets ────────────── v0.6.7
   Installed JSExpr ─────────────── v0.5.4
   Installed WebIO ──────────────── v0.8.21
```

```
   Installed FunctionalCollections ─ v0.5.0
   Installed InteractBase ────────── v0.10.10
   Installed Interact ────────────── v0.10.5
   Installed Observables ─────────── v0.5.5
   Installed CSSUtil ─────────────── v0.1.1
    Updating `C:\Users\damio\.julia\environments\v1.11\Project.toml`
  [c601a237] + Interact v0.10.5
    Updating `C:\Users\damio\.julia\environments\v1.11\Manifest.toml`
  [bf4720bc] + AssetRegistry v0.1.0
  [70588ee8] + CSSUtil v0.1.1
⌅ [3da002f7] ↓ ColorTypes v0.12.0 ⇒ v0.11.5
^ [c3611d14] ↓ ColorVectorSpace v0.11.0 ⇒ v0.10.0
⌅ [5ae59095] ↓ Colors v0.13.0 ⇒ v0.12.11
  [de31a74c] + FunctionalCollections v0.5.0
  [c601a237] + Interact v0.10.5
  [d3863d7c] + InteractBase v0.10.10
  [97c1335a] + JSExpr v0.5.4
  [bcebb21b] + Knockout v0.2.6
  [510215fc] + Observables v0.5.5
  [fa939f87] + Pidfile v1.3.0
  [0f1e0344] + WebIO v0.8.21
  [104b5d7c] + WebSockets v1.6.0
  [cc8bc4a8] + Widgets v0.6.7
  [ffd25f8a] ↑ XZ_jll v5.6.4+0 ⇒ v5.6.4+1
  [8ba89e20] + Distributed v1.11.0
        Info Packages marked with ^ and ⌅ have new versions available. Those with ^
may be upgradable, but those with ⌅ are restricted by compatibility constraints from
upgrading. To see why use `status --outdated -m`
    Building WebIO ──────→ `C:\Users\damio\.julia\scratchspaces\44cfe95a-1eb2-52ea-
b672-e2afdf69b78f\0eef0765186f7452e52236fa42ca8c9b3c11c6e3\build.log`
    Building Knockout ────→ `C:\Users\damio\.julia\scratchspaces\44cfe95a-1eb2-52ea-
b672-e2afdf69b78f\91835de56d816864f1c38fb5e3fad6eb1e741271\build.log`
    Building InteractBase → `C:\Users\damio\.julia\scratchspaces\44cfe95a-1eb2-52ea-
b672-e2afdf69b78f\aa5daeff326db0a9126a225b58ca04ae12f57259\build.log`
    Building Interact ────→ `C:\Users\damio\.julia\scratchspaces\44cfe95a-1eb2-52ea-
b672-e2afdf69b78f\c5091992248c7134af7c90554305c600d5d9012b\build.log`
Precompiling project...
    527.9 ms  ✓ Observables
    412.5 ms  ✓ Pidfile
    442.1 ms  ✓ FunctionalCollections
    566.0 ms  ✓ XZ_jll
    679.0 ms  ✓ AssetRegistry
   1404.4 ms  ✓ ColorTypes
   1305.7 ms  ✓ ColorVectorSpace
   3491.3 ms  ✓ WebSockets
    952.8 ms  ✓ ColorVectorSpace → SpecialFunctionsExt
   2467.7 ms  ✓ Colors
   1080.1 ms  ✓ Widgets
   2488.7 ms  ✓ WebIO
   1513.4 ms  ✓ CSSUtil
   1586.6 ms  ✓ JSExpr
   2009.1 ms  ✓ Knockout
   2733.9 ms  ✓ InteractBase
   1569.3 ms  ✓ Interact
  17 dependencies successfully precompiled in 16 seconds. 216 already precompiled.
  4 dependencies precompiled but different versions are currently loaded. Restart ju
```

```
lia to access the new versions. Otherwise, loading dependents of these packages may
trigger further precompilation to work with the unexpected versions.
  1 dependency had output during precompilation:
┌ WebSockets
│  WARNING: could not import Logging.termlength into WebSockets
│  WARNING: could not import Logging.showvalue into WebSockets
└
   Resolving package versions...
  No Changes to `C:\Users\damio\.julia\environments\v1.11\Project.toml`
  No Changes to `C:\Users\damio\.julia\environments\v1.11\Manifest.toml`
```

Installing the packages typically takes a moment. If you are wondering when a cell is done running, you can check on the left hand side:

In [*]: The cell is still running

In [1]: The cell is done (the number within the brackets will change).

If you want to check which packages you have installed, the following command will give you the full list and the version numbers.

In [36]:
```
Pkg.status()
```

```
Status `C:\Users\damio\.julia\environments\v1.11\Project.toml`
  [87dc4568] HiGHS v1.13.0
  [7073ff75] IJulia v1.26.0
  [c601a237] Interact v0.10.5
  [b6b21f68] Ipopt v1.7.1
  [4076af6c] JuMP v1.23.6
  [91a5bcdd] Plots v1.40.9
```

If you want to update your packages to a newer version, you can use the command `Pkg.update()` .

In [37]:
```
Pkg.update()
```

```
    Updating registry at `C:\Users\damio\.julia\registries\General.toml`
  No Changes to `C:\Users\damio\.julia\environments\v1.11\Project.toml`
  No Changes to `C:\Users\damio\.julia\environments\v1.11\Manifest.toml`
       Info We haven't cleaned this depot up for a bit, running Pkg.gc()...
     Active manifest files: 1 found
     Active artifact files: 96 found
     Active scratchspaces: 6 found
    Deleted no artifacts, repos, packages or scratchspaces
```

2.1 Building an optimization model. JuMP Variables

First, load the JuMP package into your current environment.

In [38]:
```
using JuMP
```

```
[ Info: Precompiling SpecialFunctionsExt [997ecda8-951a-5f50-90ea-61382e97704b] (cac
he misses: wrong dep version loaded (2))
```

Now you can start building your optimization model, which we will also refer to as a **JuMP model**!

Remember that there are three components to every optimization problem:

1. Decision variables (the values we are allowed to determine)
2. Objective (the goal we want to achieve, which is expressed as a function of the decision variables)
3. Constraints (limitations that describe which choices are possible to make, also expressed as functions of the decision variables)

We will go through how to model these three parts using Julia and JuMP one by one. Let's start with defining decision variables:

In [39]:
```
first_model = Model()
@variable(first_model, y >= 0)
@variable(first_model, 1 <= z <= 2)
first_model
```

Out[39]:
```
A JuMP Model
├ solver: none
├ objective_sense: FEASIBILITY_SENSE
├ num_variables: 2
├ num_constraints: 3
│ ├ VariableRef in MOI.GreaterThan{Float64}: 2
│ └ VariableRef in MOI.LessThan{Float64}: 1
└ Names registered in the model
  └ :y, :z
```

In [40]:
```
# Let's find the lower bound of the z variable
JuMP.lower_bound(z)
```

Out[40]:
```
1.0
```

Other ways to create variables

Sometimes, we need to create problems with MANY variables. Then it is useful to not have to create each variable separately.

A useful feature is that we can create arrays of JuMP variables.

In [41]:
```
model = Model()
@variable(model, x[1:4] >= 0)
x
```

Out[41]:
```
4-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]
 x[4]
```

The indices of the arrays don't have to be integers. They can be anything, like a string

"name" or a symbol :symbol_name .

Let's create a variable that uses both number indices and symbols!

In [42]:
```
model = Model()
@variable(model, x[i = 1:2, j = [:A, :B]] >= i)

println("Printing my optimization variable: ")
println()
println(x)
println()
println("The lower bound of the first element is ", JuMP.lower_bound(x[1,:A]))
```

```
Printing my optimization variable:

2-dimensional DenseAxisArray{VariableRef,2,...} with index sets:
    Dimension 1, Base.OneTo(2)
    Dimension 2, [:A, :B]
And data, a 2×2 Matrix{VariableRef}:
 x[1,A]  x[1,B]
 x[2,A]  x[2,B]

The lower bound of the first element is 1.0
```

Another example, with strings as names:

In [43]:
```
model = Model()
@variable(model, x[i = 1:4, j = ["one", "two"]] >= i)
x
```

Out[43]:
```
2-dimensional DenseAxisArray{VariableRef,2,...} with index sets:
    Dimension 1, Base.OneTo(4)
    Dimension 2, ["one", "two"]
And data, a 4×2 Matrix{VariableRef}:
 x[1,one]  x[1,two]
 x[2,one]  x[2,two]
 x[3,one]  x[3,two]
 x[4,one]  x[4,two]
```

What if I want to add two variables with the same name?

It will give an error!

In [44]:
```
model = Model()
@variable(model, x >= 1)
@variable(model, x >= 2)
```

```
An object of name x is already attached to this model. If this
    is intended, consider using the anonymous construction syntax, for example,
    `x = @variable(model, [1:N], ...)` where the name of the object does
    not appear inside the macro.

    Alternatively, use `unregister(model, :x)` to first unregister
    the existing name from the model. Note that this will not delete the
    object; it will just remove the reference at `model[:x]`.


Stacktrace:
 [1] error(s::String)
   @ Base .\error.jl:35
 [2] _error_if_cannot_register(model::Model, name::Symbol)
   @ JuMP C:\Users\damio\.julia\packages\JuMP\CU7H5\src\macros.jl:418
 [3] macro expansion
   @ C:\Users\damio\.julia\packages\JuMP\CU7H5\src\macros.jl:407 [inlined]
 [4] top-level scope
   @ In[44]:3
```

Binary and integer variables

By default, the decision variables in Julia are continuous variables. However, we can also create binary and integer variables as follows:

In [45]:
```
model = Model()
@variable(model, x >= 1, Int)
@variable(model, y, Bin)
model
```

Out[45]:
```
A JuMP Model
├ solver: none
├ objective_sense: FEASIBILITY_SENSE
├ num_variables: 2
├ num_constraints: 3
│ ├ VariableRef in MOI.GreaterThan{Float64}: 1
│ ├ VariableRef in MOI.Integer: 1
│ └ VariableRef in MOI.ZeroOne: 1
└ Names registered in the model
  └ :x, :y
```

2.2 JuMP Constraints

Now that we've seen how to create variables, let's look at **constraints**.

Remember that constraints are limitations on the valid choices of decision variables (for example, the production of football and soccer trophies is limited by the available amount of wood). They may involve one or more decision variable, and be formulated as inequalities or equalities.

Let's formulate the following constraints with decision variables $x \geq 0$ and $y \geq 0$:

$2x + y \leq 1$

$2x + y \geq 1$

$2x + y = 1$

Here is an example:

In [46]:
```
model = Model()
@variable(model, x >= 0)
@variable(model, y >= 0)

@constraint(model, c_less_than, 2x + y <= 1)
@constraint(model, c_greater_than, 2x + y >= 1)
@constraint(model, c_equal_to, 2x + y == 1)

print(model)
```

$$\begin{aligned} & \text{feasibility} \\ \text{Subject to} \quad & 2x + y = 1 \\ & 2x + y \geq 1 \\ & 2x + y \leq 1 \\ & x \geq 0 \\ & y \geq 0 \end{aligned}$$

Similar to the optimization variables, we can access the constraints using their names:

In [47]:
```
print(c_equal_to)
```

```
c_equal_to : 2 x + y == 1
```

2.3 JuMP Objective Functions

Now let's look at the last main part of the optimization model: The objective function.

Note two important aspects:

1. The objective is formulated as a function of the optimization problem.
2. We need to specify whether we want to maximize or minimize this function.

Minimization problem (i.e., minimizing the objective function):

In [48]:
```
model = Model()
@variable(model, x >= 0)

@objective(model, Min, 2x + 1)

model
```

Out[48]:
```
A JuMP Model
├ solver: none
├ objective_sense: MIN_SENSE
│ └ objective_function_type: AffExpr
├ num_variables: 1
├ num_constraints: 1
│ └ VariableRef in MOI.GreaterThan{Float64}: 1
└ Names registered in the model
  └ :x
```

Maximization problem (i.e., maximizing the objective function):

In [49]:
```
model = Model()
@variable(model, x <= 2)

@objective(model, Max, 2x + 1)

model
```

Out[49]:
```
A JuMP Model
├ solver: none
├ objective_sense: MAX_SENSE
│ └ objective_function_type: AffExpr
├ num_variables: 1
├ num_constraints: 1
│ └ VariableRef in MOI.LessThan{Float64}: 1
└ Names registered in the model
  └ :x
```

2.4 DO IT YOURSELF!

Try to build the optimization model.

Top Brass Trophy Company makes large championship trophies for youth athletic leagues. At the moment, they are planning production for fall sports: football and soccer. Each football trophy has a wood base, an engraved plaque, a large brass football on top, and returns 12 dollars in profit. Soccer trophies are similar except that a brass soccer ball is on top, and the unit profit is only 9 dollars. Since the football has an asymmetric shape, its base requires 4 board feet of wood; the soccer base requires only 2 board feet. At the moment there are 1000 brass footballs in stock, 1500 soccer balls, 1750 plaques, and 4800 board feet of wood. What trophies should be produced from these supplies to maximize total profit assuming that all that are made can be sold?

(This is not a graded exercise for correctness, but just to see if you can start working with the syntax) There is a video available on the Canvas course website -- in the Julia/JuMP Resources page -- where you can see how to complete this.

In [51]:
```
# Top Brass Optimization Model

TBmodel = Model()
@variable(TBmodel, football_trophies >= 0)                    # Number of
```

```
@variable(TBmodel, soccer_trophies >= 0)                                # Number of

@constraint(TBmodel, football_trophies <= 1000)                         # Max 1000 b
@constraint(TBmodel, soccer_trophies <= 1500)                           # Max 1500 b
@constraint(TBmodel, football_trophies + soccer_trophies <= 1750)       # Max 1750 p
@constraint(TBmodel, 4*football_trophies + 2*soccer_trophies <= 4800)    # Max 4800

@objective(TBmodel, Max, 12*football_trophies + 9soccer_trophies)

TBmodel
```

Out[51]:
```
A JuMP Model
├ solver: none
├ objective_sense: MAX_SENSE
│ └ objective_function_type: AffExpr
├ num_variables: 2
├ num_constraints: 6
│ ├ AffExpr in MOI.LessThan{Float64}: 4
│ └ VariableRef in MOI.GreaterThan{Float64}: 2
└ Names registered in the model
  └ :football_trophies, :soccer_trophies
```

2.5 Solving a Model

Once we've formulated a model, the next step is to solve it. This requires a **solver**.

Solvers are implementations of algorithms in software that are designed to We will talk about solvers in the class on Tuesday, January 28. If you do this tutorial before that and feel confused, you can stop here. Otherwise, think of a solver as an amazing piece of software that will mtake your optimization model as an input and magically give you the solution back! (In fact, formost of the class, this is how we will think about solvers anyways.)

JuMP supports lots of different solvers. The JuMP documentation contains a list of the supported solvers and the types of problems each solver supports.

For this tutorial, we're going to use two solvers in particular.

The first solver is HiGHS.

The second solver is the COIN-OR Interior Point OPTimizer (Ipopt). This solver supports nonlinear programs with continous variables.

Ipopt is available via the Ipopt.jl package.

You may see other solvers in the class from time to time, such as Clp, GLPK, Gurobi. We will talk more about differences between the solvers as the class progresses.

In [52]:
```
using HiGHS
using Ipopt
```

There are two ways to add a solver to a JuMP model:

In [53]:
```
model = Model(HiGHS.Optimizer)

# ... or ...

model = Model()
set_optimizer(model, HiGHS.Optimizer)
```

Each solver can only handle certain types of problems. If you try to solve an unsupported problem type (such as using the HiGHS solver for a highly nonlinear problem), an error will be thrown:

In [57]:
```
model = Model(HiGHS.Optimizer)
@variable(model, 0 <= x <= π)
@NLobjective(model, Min, cos(x)^2) # NOTE: This is a non-linear objective function!
optimize!(model)
```

```
Running HiGHS 1.9.0 (git hash: 66f735e60): Copyright (c) 2024 HiGHS under MIT licenc
e terms
The solver does not support nonlinear problems (that is, NLobjective and NLconstrain
t).

Stacktrace:
 [1] error(s::String)
   @ Base .\error.jl:35
 [2] optimize!(model::Model; ignore_optimize_hook::Bool, _differentiation_backend::M
athOptInterface.Nonlinear.SparseReverseMode, kwargs::@Kwargs{})
   @ JuMP C:\Users\damio\.julia\packages\JuMP\CU7H5\src\optimizer_interface.jl:601
 [3] optimize!(model::Model)
   @ JuMP C:\Users\damio\.julia\packages\JuMP\CU7H5\src\optimizer_interface.jl:546
 [4] top-level scope
   @ In[57]:4
```

Let's try instead if the IpOpt solver, which is a solver for non-linear optimization problems.

In [58]:
```
set_optimizer(model, Ipopt.Optimizer)
optimize!(model)
```

```
This is Ipopt version 3.14.17, running with linear solver MUMPS 5.7.3.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        1

Total number of variables............................:        1
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        1
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  9.9990000e-01 0.00e+00 2.00e-02  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  9.8759980e-01 0.00e+00 3.07e-02  -1.0 1.02e-01    -  8.54e-01 1.00e+00f  1
   2  9.8669625e-01 0.00e+00 4.01e-01  -1.7 4.01e-03   2.0 1.00e+00 1.00e+00f  1
   3  9.8380423e-01 0.00e+00 4.00e-01  -1.7 1.20e-02   1.5 1.00e+00 1.00e+00f  1
   4  9.7257905e-01 0.00e+00 4.30e-01  -1.7 3.88e-02   1.0 1.00e+00 1.00e+00f  1
   5  8.8618271e-01 0.00e+00 6.31e-01  -1.7 1.78e-01   0.6 1.00e+00 1.00e+00f  1
   6  8.2896451e-01 0.00e+00 8.03e-01  -1.7 8.22e-02   1.0 1.00e+00 1.00e+00f  1
   7  4.8405038e-01 0.00e+00 9.92e-01  -1.7 3.75e-01   0.5 9.32e-01 1.00e+00f  1
   8  1.4382245e-01 0.00e+00 7.49e-01  -1.7 1.49e+01    -  1.00e+00 7.76e-02f  2
   9  7.8035855e-03 0.00e+00 1.89e-01  -1.7 4.77e-01    -  8.77e-01 1.00e+00f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10  3.6166261e-07 0.00e+00 1.86e-03  -2.5 8.91e-02    -  1.00e+00 1.00e+00f  1
  11  2.5240578e-13 0.00e+00 5.79e-10  -3.8 6.01e-04    -  1.00e+00 1.00e+00f  1
  12  9.1318054e-22 0.00e+00 1.57e-16  -5.7 5.02e-07    -  1.00e+00 1.00e+00f  1
  13  3.7493995e-33 0.00e+00 1.89e-16  -8.6 3.02e-11    -  1.00e+00 1.00e+00f  1

Number of Iterations....: 13

                                   (scaled)                 (unscaled)
Objective...............:   3.7493994566546440e-33    3.7493994566546440e-33
Dual infeasibility......:   1.8934343678172697e-16    1.8934343678172697e-16
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5059035951250753e-09    2.5059035951250753e-09
Overall NLP error.......:   2.5059035951250753e-09    2.5059035951250753e-09


Number of objective function evaluations             = 19
Number of objective gradient evaluations             = 14
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 13
Total seconds in IPOPT                               = 0.003

EXIT: Optimal Solution Found.
```

2.6 Getting solutions

- Use `objective_value(::Model)` to get the objective value
- Use `value(::VariableRef)` to get the value of a variable

In [59]:
```
x_value = value(x)
obj_value = objective_value(model)

println("The optimal decision is ", x_value)
println("The objective value is ", obj_value)
```

```
The optimal decision is 1.5707963267948966
The objective value is 3.749399456654644e-33
```

More advanced - to remember for later in class!

After solving a model, JuMP can report three different statuses:

- `termination_status(model)` explains why the solver stopped. Common statuses are `OPTIMAL`, `INFEASIBLE`, `DUAL_INFEASIBLE` (i.e., primal is potentially unbounded), and `LOCALLY_SOLVED`.
- `primal_status(model)` explains how to interpret the primal solution vector. Common statuses are `FEASIBLE_POINT` and `NO_SOLUTION`.
- `dual_status(model)` explains how to interpret the dual solution vector. Common statuses are `FEASIBLE_POINT` and `NO_SOLUTION`.

Information about both primal and dual variables are available:

- Use `value(::VariableRef)` to get the value of a primal variable
- Use `dual(::ConstraintRef)` to get the value of the dual variable associated with a constraint

In [60]:
```
model = Model(optimizer_with_attributes(Ipopt.Optimizer))
@variable(model, 0 <= x <= π)
@NLobjective(model, Min, cos(x)^2) # NOTE: This is a non-linear objetive function!
optimize!(model)
x_value = value(x)
obj_value = objective_value(model)

println()
println("====== Let's look at the solution! =======")
println()
println("Termination status: ", termination_status(model))
println("Primal status:      ", primal_status(model))
println("Dual status:        ", dual_status(model))
println("      x | $(x_value)")
println("    π/2 | $(π/2)")
println("--------+----------------------")
println("cos²(x) | $(obj_value)")
```

```
This is Ipopt version 3.14.17, running with linear solver MUMPS 5.7.3.

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        1

Total number of variables............................:        1
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        1
                     variables with only upper bounds:        0
Total number of equality constraints.................:        0
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  9.9990000e-01 0.00e+00 2.00e-02  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  9.8759980e-01 0.00e+00 3.07e-02  -1.0 1.02e-01    -  8.54e-01 1.00e+00f  1
   2  9.8669625e-01 0.00e+00 4.01e-01  -1.7 4.01e-03   2.0 1.00e+00 1.00e+00f  1
   3  9.8380423e-01 0.00e+00 4.00e-01  -1.7 1.20e-02   1.5 1.00e+00 1.00e+00f  1
   4  9.7257905e-01 0.00e+00 4.30e-01  -1.7 3.88e-02   1.0 1.00e+00 1.00e+00f  1
   5  8.8618271e-01 0.00e+00 6.31e-01  -1.7 1.78e-01   0.6 1.00e+00 1.00e+00f  1
   6  8.2896451e-01 0.00e+00 8.03e-01  -1.7 8.22e-02   1.0 1.00e+00 1.00e+00f  1
   7  4.8405038e-01 0.00e+00 9.92e-01  -1.7 3.75e-01   0.5 9.32e-01 1.00e+00f  1
   8  1.4382245e-01 0.00e+00 7.49e-01  -1.7 1.49e+01    -  1.00e+00 7.76e-02f  2
   9  7.8035855e-03 0.00e+00 1.89e-01  -1.7 4.77e-01    -  8.77e-01 1.00e+00f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10  3.6166261e-07 0.00e+00 1.86e-03  -2.5 8.91e-02    -  1.00e+00 1.00e+00f  1
  11  2.5240578e-13 0.00e+00 5.79e-10  -3.8 6.01e-04    -  1.00e+00 1.00e+00f  1
  12  9.1318054e-22 0.00e+00 1.57e-16  -5.7 5.02e-07    -  1.00e+00 1.00e+00f  1
  13  3.7493995e-33 0.00e+00 1.89e-16  -8.6 3.02e-11    -  1.00e+00 1.00e+00f  1

Number of Iterations....: 13

                                   (scaled)                 (unscaled)
Objective...............:   3.7493994566546440e-33    3.7493994566546440e-33
Dual infeasibility......:   1.8934343678172697e-16    1.8934343678172697e-16
Constraint violation....:   0.0000000000000000e+00    0.0000000000000000e+00
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   2.5059035951250753e-09    2.5059035951250753e-09
Overall NLP error.......:   2.5059035951250753e-09    2.5059035951250753e-09


Number of objective function evaluations             = 19
Number of objective gradient evaluations             = 14
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 13
Total seconds in IPOPT                               = 0.003

EXIT: Optimal Solution Found.

====== Let's look at the solution! =======
```

```
Termination status: LOCALLY_SOLVED
Primal status:      FEASIBLE_POINT
Dual status:        FEASIBLE_POINT
      x | 1.5707963267948966
    π/2 | 1.5707963267948966
--------+----------------------
cos²(x) | 3.749399456654644e-33
```

Interpreting statuses

After solving a model, JuMP can report three different statuses:

- `termination_status(model)` explains why the solver stopped. Common statuses are `OPTIMAL`, `INFEASIBLE`, `DUAL_INFEASIBLE` (i.e., primal is potentially unbounded), and `LOCALLY_SOLVED`.
- `primal_status(model)` explains how to interpret the primal solution vector. Common statuses are `FEASIBLE_POINT` and `NO_SOLUTION`.
- `dual_status(model)` explains how to interpret the dual solution vector. Common statuses are `FEASIBLE_POINT` and `NO_SOLUTION`.

Information about both primal and dual variables are available:

- Use `value(::VariableRef)` to get the value of a primal variable
- Use `dual(::ConstraintRef)` to get the value of the dual variable associated with a constraint