Linear(ish) programs in Julia+JuMP

15.S60 Computing in Optimization and Statistics, Session 6

MIT IAP 2022

Learning Objectives

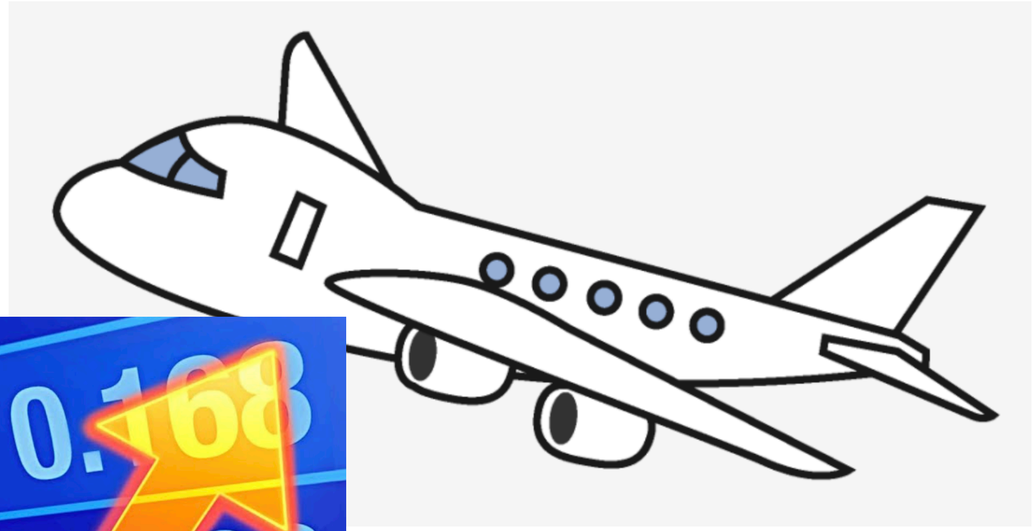
By the end of this session you will be able to…

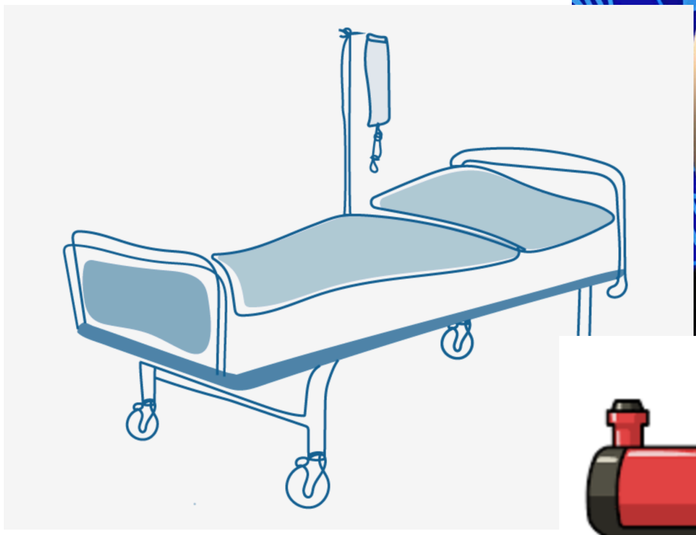
- Implement code in Julia with standard packages
- Write mathematical programs in Julia with JuMP using diverse model inputs
- Design and code data structures to hold optimization model inputs in Julia
- Code extensible, scalable optimization pipelines using modularization

Agenda

- **Introduction** — Julia syntax
- **JuMP basics** — build and solve a simple mathematical program
- **Extended case study** — design an optimization workflow in Julia moving from raw data to optimal decisions

You already know optimization is important and cool





Code flow != workflow

script.jl

```
Pre-process raw data

Format model inputs

Build and solve model
```

Here is my script.

How did I design it?

Code flow != workflow

script.jl

```
Pre-process raw data

Format model inputs

Build and solve model
```

Step 1: First, I formulated and wrote my model.

Code flow != workflow

script.jl

```
Pre-process raw data

Format model inputs

Build and solve model
```

Step 2: Then I formatted the model components (sets, parameters…) with available raw data in mind

Code flow != workflow

script.jl

```
Pre-process raw data

Format model inputs

Build and solve model
```

Step 3: Then I decide what raw data I need for downstream model inputs.

Code flow != workflow

script.jl

```
Pre-process raw data

Format model inputs

Build and solve model
```

JuMP will help you build and solve a model.

It is equally important to have an organized process for all of the other steps!

How to stay organized??

But first things first…

1. Julia basics!

2. JuMP basics + useful code modules

3. Piecing together those modules into an optimization pipeline

Tools for staying organized

Directories and scripts are your friend

```
src
└─model
   └─ data_structure.jl
   └─ model1.jl
   └─ model2.jl
└─pipeline
   └─ script.jl

└─data
   └─ preprocess.jl
   └─ postprocess.jl
test
└─unit_test_and_benchmark.jl
```

Tools for staying organized

Directories and scripts are your friend

```
src
└─model
   └─ data_structure.jl
   └─ model1.jl
   └─ model2.jl
└─pipeline
   └─ script.jl
└─data
   └─ preprocess.jl
   └─ postprocess.jl
test
└─unit_test_and_benchmark.jl
```

script.jl

```
Using packages

include ../model/*


D = DataStructure

m = model1(D)

solve_model(m)
```

Tools for staying organized

Directories and scripts are your friend

```
src
└─model
  └─ data_structure.jl
  └─ model1.jl
  └─ model2.jl
└─pipeline
  └─ script.jl
└─data
  └─ preprocess.jl
  └─ postprocess.jl
test
└─unit_test_and_benchmark.jl
```

data_str…

```
struct DataStructure
    attr1
    attr2
end

function DataStructure(
    data_direc
)
    constructor stuff

    D = DataStructure(
    attr1, attr2)

    return D
end
```

Tools for staying organized

Directories and scripts are your friend

```
src
└─model
    └─ data_structure.jl
    └─ model1.jl
    └─ model2.jl
└─pipeline
    └─ script.jl
└─data
    └─ preprocess.jl
    └─ postprocess.jl
test
└─unit_test_and_benchmark.jl
```

model1.jl

function to build model1

function(s) to solve model1

What else?