



Escuela de Primavera en  
**Investigación de Operaciones**

28 al 30 de abril de 2025

Modalidad Virtual

# “Modelado Matemático Moderno: Caso Práctico con Julia”

Parte 1

Presentación por  
Eduardo Salazar Treviño

# Contenido

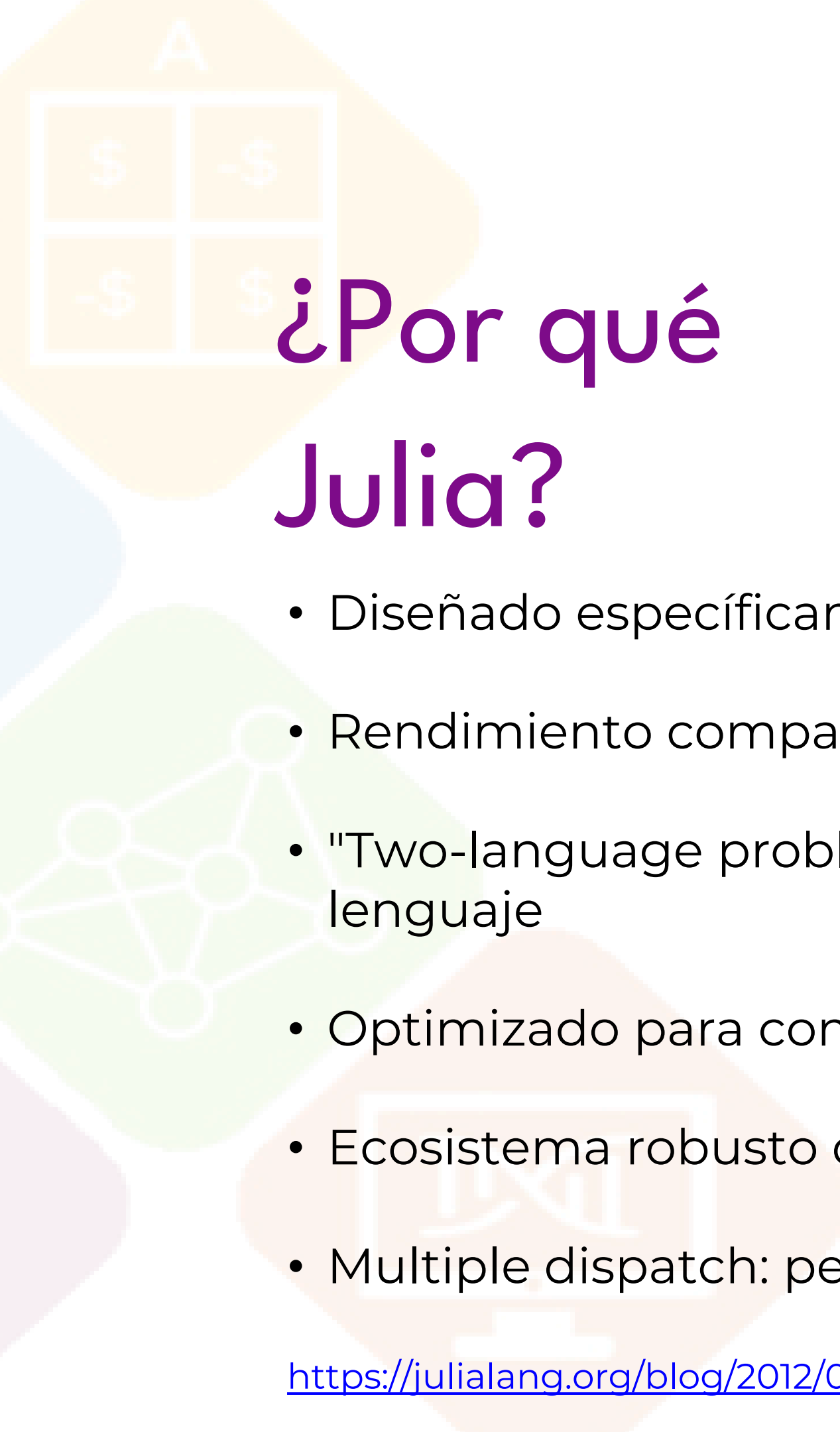
- Sesión 1: Fundamentos de Julia y JuMP
  - Introducción a Julia:
    - Sintaxis básica
    - Funciones y múltiple dispatch
    - Arrays, operaciones vectorizadas y broadcasting
    - Comparación con Python/gurobipy
    - Gestión de paquetes con Pkg y su uso
    - Performance: Compilación JIT, Funciones y Tipos Inestables
  - Introducción a JuMP
    - Estructura básica de un modelo
    - Variables, restricciones y función objetivo
    - Conexión con solvers (Gurobi, CPLEX, solvers open source)



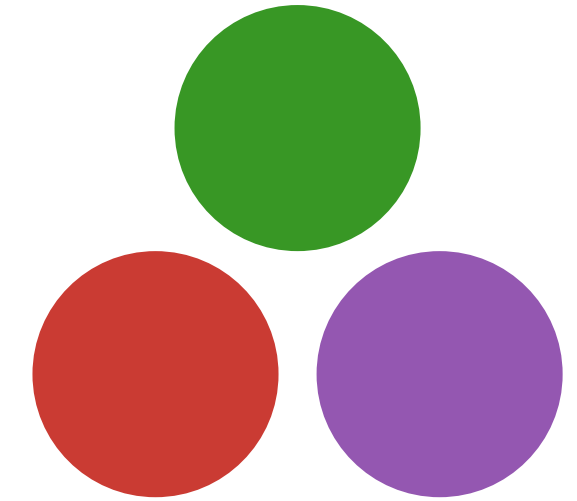
# Requisitos Previos

- ✓= Tener instalado Julia (<https://julialang.org/install/>)
- ✓= Tener instalado Visual Studio Code (NO Visual Studio)  
(<https://code.visualstudio.com/>)
- ✓= Solver de Optimización: Gurobi, CPLEX, HiGHS

# Introducción a Julia



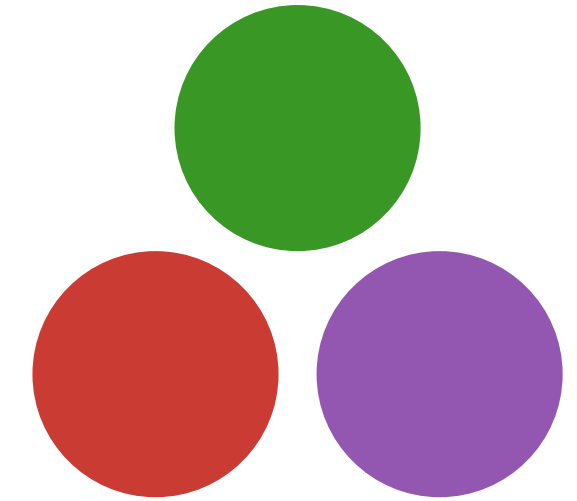
# ¿Por qué Julia?



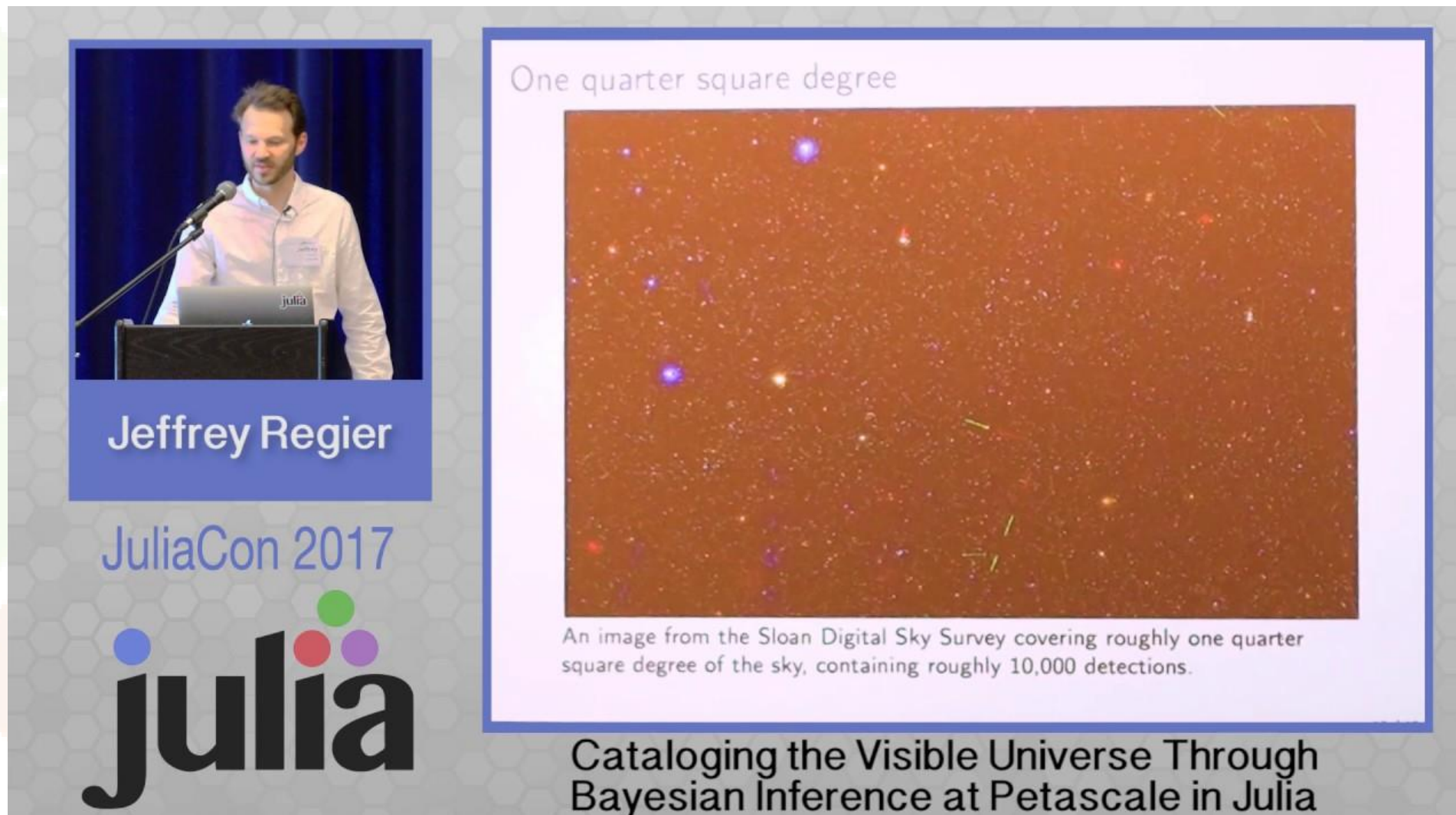
- Diseñado específicamente para computación científica
- Rendimiento comparable a C/C++ con sintaxis de alto nivel como Python
- "Two-language problem" resuelto: prototipado y producción en el mismo lenguaje
- Optimizado para computación numérica y álgebra lineal
- Ecosistema robusto de paquetes científicos
- Multiple dispatch: permite paradigmas de programación flexibles

<https://julialang.org/blog/2012/02/why-we-created-julia/>

# ¿Por qué Julia?



[Celeste.jl:](#)  
[Petascale](#)  
[Computing in](#)  
[Julia | Prabhat,](#)  
[Regier & Fischer](#)  
[| JuliaCon 2017](#)



The slide features a photograph of Jeffrey Regier at a podium on the left. To his right is a large astronomical image of a star field. Below the photo is the text 'Jeffrey Regier'. Below that is 'JuliaCon 2017' and the Julia logo. The astronomical image is titled 'One quarter square degree' and has a caption: 'An image from the Sloan Digital Sky Survey covering roughly one quarter square degree of the sky, containing roughly 10,000 detections.' At the bottom of the slide is the title 'Cataloging the Visible Universe Through Bayesian Inference at Petascale in Julia'.

Jeffrey Regier

JuliaCon 2017

One quarter square degree

An image from the Sloan Digital Sky Survey covering roughly one quarter square degree of the sky, containing roughly 10,000 detections.

Cataloging the Visible Universe Through Bayesian Inference at Petascale in Julia



# Tipos de Datos

```
x = 10           # Int64
y = 3.14         # Float64
z = "Hola"      # String
b = true        # Bool
dospi = 2π
explicito = 1::Int64
```



# Estructuras de Control: Condicionales



```
if x > 5
    println("x es mayor que 5")
elseif x < 0
    println("x es negativo")
else
    println("x está entre 0 y 5")
end
```

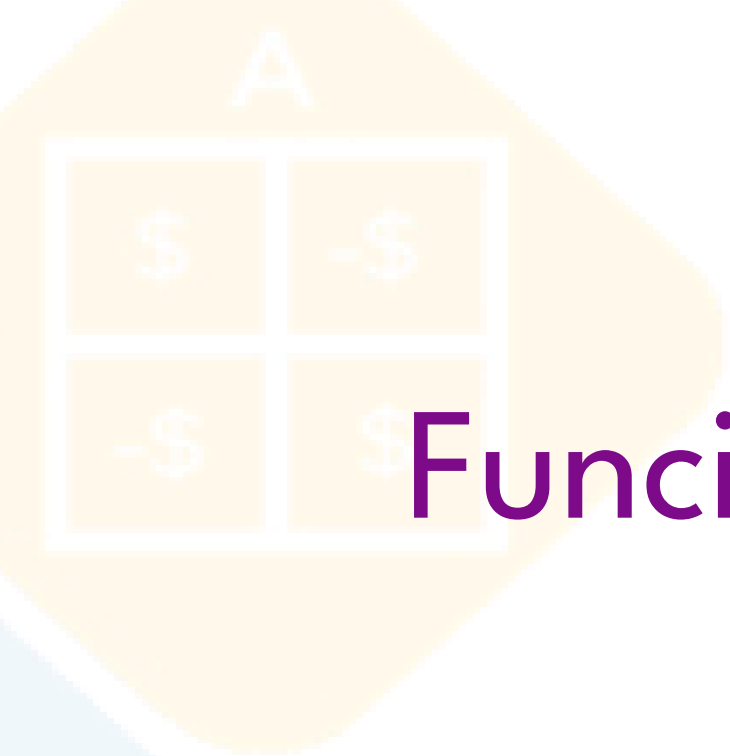




# Estructuras de Control: Bucles

```
for i in 1:5
    println(i^2)
end

i = 1
while i <= 5
    println(i)
    i += 1
end
```



# Funciones Simples



```
function suma(a, b)  
    return a + b  
end
```

```
suma_concisa(a, b) = a + b
```



# Multiple Dispatch



```
function procesar(x::Int64)
    println("Procesando entero: $x")
end

function procesar(x::String)
    println("Procesando texto: $x")
end

procesar(42)          # Llama a la primera versión
procesar("Julia")     # Llama a la segunda versión
```

# Funciones anónimas, mapas



```
f = x -> x^2 + 2x - 1  
f(2)  
map(f, [1, 2, 3, 4])
```



¿Vamos bien?



# Arrays, acceso y slicing



```
a = [1, 2, 3, 4, 5]
b = zeros(3, 3)      # Matriz 3x3 de ceros
c = ones(2, 2, 2)    # Tensor 3D de unos
d = rand(5)          # Vector aleatorio

# Acceso a elementos
a[1]                  # Primer elemento (Index en 1!)
b[2, 3]               # Elemento en fila 2, columna 3

# Slicing
a[2:4]                # [2, 3, 4]
b[1:2, :]             # Primeras dos filas
```



# Broadcasting y comprensiones



```
a .+ 10          # Suma 10 a cada elemento
a .* a          # Eleva al cuadrado cada elemento

# Comprensiones
[i^2 for i in 1:10]
[i*j for i in 1:3, j in 1:3] # Matriz 3x3
```



¿Seguimos bien?



# Instalación de Paquetes y Entornos

Análogo a `pip` en Python, Julia cuenta con `Pkg` que permite instalar paquetes y manejar entornos aislados para cada proyecto.

```
julia>
```

```
(epio) pkg>
```

Podemos acceder a `Pkg` en la línea de comandos de Julia tecleando ]

```
help?>
```

```
(epio) pkg> add Gurobi, Plots
```

Las dependencias en Julia se describen a través de un archivo llamado `Project.toml`

Los entornos nos permiten manejar de manera aislada y controlada las dependencias de distintos proyectos



# Usando Paquetes: CSV y DataFrames

```
using CSV, DataFrames
data = CSV.read("datos.csv", DataFrame)

# Escritura a un archivo
open("resultado.txt", "w") do file
    write(file, "Resultado del análisis\n")
    write(file, "Valor óptimo: 42.0\n")
end

# Con DataFrames
df = DataFrame(x = 1:5, y = rand(5))
CSV.write("output.csv", df)
```

# Usando Paquetes: Plots



using Plots

# Gráfico simple

x = 0:0.1:10

y = sin.(x)

plot(x, y, label="sen(x)", title="Función seno", xlabel="x",  
 ylabel="sen(x)")



# Rendimiento: JIT y Funciones

```
using BenchmarkTools

# Versión con variables globales
function test_global()
    global x = 0
    for i in 1:10_000_000
        x += i
    end
    return x
end

# Versión con función
function test_function()
    function suma_local()
        x = 0
        for i in 1:10_000_000
            x += i
        end
        return x
    end
    return suma_local()
end

# Medir rendimiento
println("Versión con variables globales:")
@btime test_global()

println("\nVersión con función:")
@btime test_function()
```

Julia es compilado Just in Time:  
La primera vez que se mande llamar una función se compilará.  
Siempre es mejor para el rendimiento tener todo en funciones.  
**EVITAR** el uso de variables globales y tipos inestables.



# Rendimiento: Tipos Inestables

```
function suma_estable(n)
    suma = 0
    for i in 1:n
        suma += i # Siempre suma enteros
    end
    return suma
end
```

```
function suma_inestable(n)
    suma = 0
    for i in 1:n
        if i % 2 == 0
            suma += i # Suma un Int
        else
            suma += i/2.0 # Suma un Float - ¡Tipo inestable!
        end
    end
    return suma
end
```

# Rendimiento: Tipos Inestables

```
function suma_anotada(n)
    suma::Float64 = 0.0 # Anotación de tipo explícita
    for i in 1:n
        if i % 2 == 0
            suma += i      # Int convertido automáticamente a Float64
        else
            suma += i/2.0  # Float64
        end
    end
    return suma
end
```

```
julia> @btime suma_estable(10_000_000)
2.600 ns (0 allocations: 0 bytes)
50000005000000

julia> @btime suma_inestable(10_000_000)
21.861 ms (0 allocations: 0 bytes)
3.7500005e13

julia> @btime suma_anotada(10_000_000)
6.641 ms (0 allocations: 0 bytes)
3.7500005e13
```



¿Todo bien?

# Introducción a JuMP



# ¿Qué es JuMP?

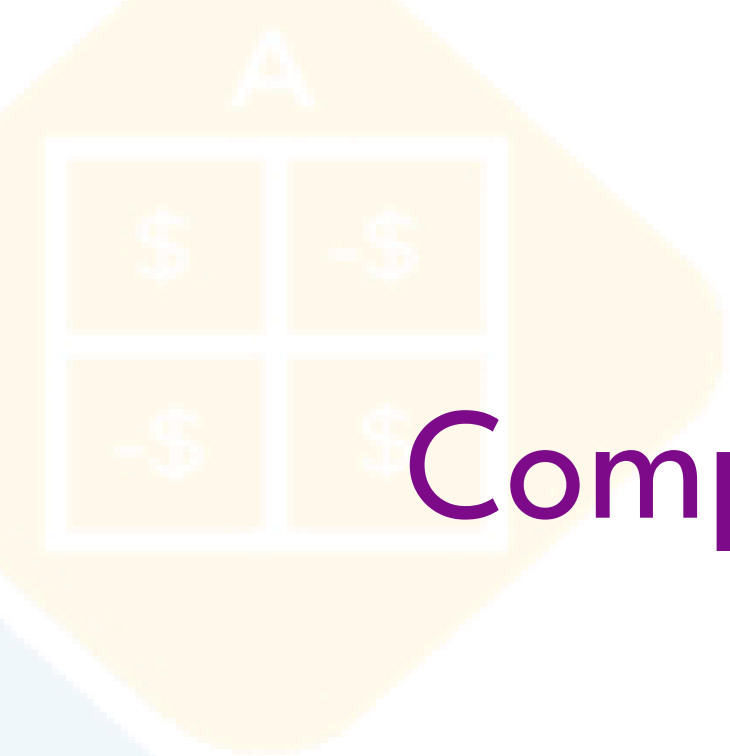


- JuMP = Julia for Mathematical Programming
- DSL (Domain Specific Language) para modelado matemático (pyomo, AMPL)
- Interfaz unificada para múltiples solvers: Gurobi, CPLEX, Xpress, HiGHS, Cbc...
- Sintaxis declarativa cercana a la notación matemática
- Alto rendimiento con flexibilidad

<https://jump.dev/>



# Comparativa con Python



```
using JuMP
using Gurobi
```

```
model = Model(Gurobi.Optimizer)
@variable(model, x >= 0)
@variable(model, y >= 0)
@constraint(model, 2x + y <= 10)
@objective(model, Max, x + y)
optimize!(model)
```



```
import gurobipy as gp
from gurobipy import GRB

model = gp.Model()
x = model.addVar(lb=0, name="x")
y = model.addVar(lb=0, name="y")
model.addConstr(2*x + y <= 10)
model.setObjective(x + y, GRB.MAXIMIZE)
model.optimize()
```



# Ejemplo Básico

$$\max 5x + 3y - 2z \quad (1)$$

s.t.

$$2x + y \leq 10 \quad (2)$$

$$x + 5y \geq 15 \quad (3)$$

$$x \geq 0 \quad (4)$$

$$0 \leq y \leq 10 \quad (5)$$

$$z \in 0, 1 \quad (6)$$



# Ejemplo Básico

```
using JuMP, Gurobi

model = Model(Gurobi.Optimizer)

# Definir variables
@variable(model, x >= 0) # Restricción 4
@variable(model, 0 <= y <= 10) # Restricción 5
@variable(model, z, Bin) # Variable binaria, Restricción 6

# Definir restricciones
@constraint(model, con1, 2x + y <= 10) # Restricción 2
@constraint(model, con2, x + 5y >= 15) # Restricción 3, usamos \geq

# Definir función objetivo
@objective(model, Max, 5x + 3y - 2z)

# Resolver el modelo
optimize!(model)

# Verificar estado de la solución
status = termination_status(model)
```



# Tipos de Variables

```
# Variables continuas
@variable(model, x)           # Sin límites
@variable(model, y >= 0)      # Límite inferior
@variable(model, 0 <= z <= 10) # Ambos límites

# Variables enteras
@variable(model, p, Int)      # Entera
@variable(model, q >= 0, Int) # Entera no negativa

# Variables binarias
@variable(model, t, Bin)      # Binaria (0-1)

# Vectores/matrices de variables
@variable(model, a[1:5])      # Vector de 5 variables
@variable(model, b[1:3, 1:4]) # Matriz 3x4 de variables
@variable(model, c[1:5] >= 0, Int) # Vector de enteros no negativos
```



# Tipos de Restricciones

```
# Restricciones lineales
@constraint(model, 2x + 3y <= 10)

# Restricciones de igualdad
@constraint(model, x + y == 5)

# Múltiples restricciones
@constraint(model, [i=1:3], x[i] + y[i] <= 10)

# Restricciones cuadráticas (con solvers que lo soporten)
@constraint(model, x^2 + y^2 <= 1)

# Restricciones no lineales
@constraint(model, sin(x) + exp(y) <= 5) # Antes NLConstraint

# Restricciones de conjunto
@constraint(model, [x, y] in SecondOrderCone())
```



# Tipos de Función Objetivo

```
● ● ●  
  
# Maximización  
@objective(model, Max, 5x + 3y)  
  
# Minimización  
@objective(model, Min, 2x^2 + y^2 - x*y)  
  
# Objetivo no lineal  
@objective(model, Min, sin(x)^2 + log(1+y))
```



# Parámetros de Solver

```
# Para Gurobi
model = Model(Gurobi.Optimizer)
set_optimizer_attribute(model, "TimeLimit", 60) # Límite de 60 segundos
set_optimizer_attribute(model, "MIPGap", 0.01) # Gap relativo del 1%
set_optimizer_attribute(model, "OutputFlag", 0) # Desactivar output

# Para CPLEX
model = Model(CPLEX.Optimizer)
set_optimizer_attribute(model, "CPXPARAM_TimeLimit", 60)
set_optimizer_attribute(model, "CPXPARAM_MIP_Tolerances_MIPGap", 0.01)
```



# Problema de la Mochila

```
function knapsack()  
    # Maximizar  $\sum_{i=1:n} v_i x_i$   
    # s.a.  $\sum_{i=1:n} w_i x_i \leq W$ ,  $x_i \in \{0,1\}$   
    v = [5, 3, 2, 7, 4] # Valores, vector columna  
    w = [2, 8, 4, 2, 5] # Pesos, vector columna  
    W = 10 # Capacidad  
    model = Model(Gurobi.Optimizer)  
    @variable(model, x[1:5], Bin) # Vector columna  
    @objective(model, Max, v' * x) # La transpuesta de v por las variables de decisión  
    @constraint(model, w' * x <= W) # La transpuesta de w por las variables de decisión  
    optimize!(model)  
    # Mostrar resultados  
    if termination_status(model) == OPTIMAL  
        println("Valor óptimo: ", objective_value(model))  
        println("Items seleccionados: ", findall(i -> value(x[i]) > 0.9, 1:length(v)))  
    end  
end
```



# Sesión de Preguntas

Gracias...