```
1 using PlutoUI,Images
```

Presentación

Programación en Julia: Herramienta para la enseñanza

Julia Crash Course

Benjamín Pérez

Héctor Medel

Empezando con VSCode

- Podemos abrir VSCode haciendo click sobre el ícono de la aplicación

- Abrimos un nuevo archivo de Julia

- Seleccione el lenguaje: Julia en este caso

- Iniciar REPL

`Shift + Enter`

Algunos atajos

Sobre el editor

- Ejecutar código en REPL y moverse: `Shift+Enter`
- Ejecutar código en REPL: `Ctrl+Enter`
- Ejecutar código de una celda en REPL: `Alt+Enter`
- Ejecutar código de una celda en REPL y moverse: `Alt+Shift+Enter`
- Limpiar resultados de la línea Clear Inline Results In Editor: Ctrl+I Ctrl+C

Sobre Terminal

- Interrumpir Ejecución: Ctrl+C
- Limpiar los resultados de la línea actual: `Escape`
- Buscar en el historial: `Ctrl+r`
- Moverse en el historial: `arriba o abajo`
- Salir: `Ctrl+d`

Correr código de un script

Podemos escribir código de `Julia` en el editor. Los comandos se autocompletarán.

Ejemplo: Hola mundo

```
print("Hola mundo!")
```

- Salvemos el archivo con algún nombre, por ejemplo: `Hola1.jl`
- Corramos el código en REPL

```
include("Hola1.jl")
```

Símbolos matemáticos

Podemos usar símbolos de LaTeX

Ejemplo: Símbolos griegos

```
θ = π/3
```

```
\theta+Tab
\pi+Tab
```

Variables y tipos

Julia es un lenguaje de tipado opcional; se puede escribir un código sin mencionar el tipo de variable.

Ejemplo:

```
a = 1 + 1;
typeof(a) # Int64

a = "hola"
typeof(a) # String
```

Jerarquía de tipos

Conversion de enteros a flotantes

```
Float64(2) # 2.0 (double-precision)
Float32(2) # 2.0f0 (single-precision)
Float16(2) # Float16(2.0) (half-precision)
```

Conversión de flotante a entero

```
Int64(2.0)        # 2
Int64(2.4)        # Error!
floot(Int64,2.4)  # 2
ceil(Int64,2.4)   # 3
round(Int64,2.4)  # 2
```

División

Existen tres operaciones asociadas

- División normal `/`
- División truncada a un entero `div(x,y)`
- El resto de una división `rem(x,y)`

Ejemplos

```
a = 1/2  # 0.5 (Float 64)

div(10,3) # 3
÷(10,3)   # 3

rem(10,3)   # 1
10%3        #1
```

Variables Booleanas

Las variables boolenas son `true` y `false`.

Operadores lógicos

Los operadores lógicos son

Operador	Sintáxis	Descripción
not	!x	regresa el opuesto de x
and	x&&y	regresa true si los dos operadores son ciertos
or	x || y	regresa true si al menos uno de los operandos es cierto

Booleano a número

Como Bool es un subtipo de `Entero`, entonces, `true` regresa `1` y `false` regresa `0`

Ejemplo:

```
true + true # 2
```

Strings

Julia tiene un buen manejo de strings

Ejemplo: Strings y chars

```
"Hola"    # string
'H'       # char
'Hola'    # Error
```

Concatenación

Ejmplo:

```
"Hola" * "mundo"            # "Holamundo"
string("Hola"," ","mundo")  # "Hola mundo"
```

Convertir un número a string

Hay varias formas de convertir un número a string

Ejemplo:

```
string(1/7)             #Regresa "0.14285714285714285"
"$(1/7)"                #Regresa "0.14285714285714285"


using Printf
@sprintf("%6.4f",1/7)  #Regresa "0.1429"
```

Función `string`

Ejemplo

```
string("podemos concanetar una string a un número ",5/6) # "podemos concanetar un
a string a un número 0.8333333333333334"
```

Interpolación

```
"podemos concanetar una string a un número $(5/6)" # "podemos concanetar una stri
ng a un número 0.8333333333333334"
```

Bucles

Bucle simple `for`

Ejemplo

```
x = 1:10;

for i in x
 println(i)
end
```

Ejemplo:

```
x = 0;
for k in 1:1000
 x = x + (1/k)^2
end
```

Bucles anidados

```
for i in 1:3
    for j in 1:3
        println("i = ", i, " j = ",j, "\n")
    end
end
```

Otra forma equivalente es:

```
for i in 1:3, j  in 1:3
    println("i = ", i, " j = ",j, "\n")
end
```

Comandos `break` y `continue`

Sirven para romper un bucle o para hacer ciertas acciones sin considerar algunos valores

Ejemplo 1

```
x=0
for k in 1:100000
    term = (1/k)^2
    x = x + term
    if (abs(term) < 1e-10)
        break
    end
end
```

Ejemplo

Lo anterior también lo podemos escribir con un bucle condicionado `while`

```
x=0
iter = 0
while ( iter == 0 || abs(term) < 1e-10) && (iter < 100000)
    term = (1/k)^2
    x = x + term
    iter = iter + 1
end
```

`continue`

Si queremos saltar una (o varias) repeticiones dentro de un loop podemos usar el comando `continue` .

Ejemplo

```
    for n in 1:10
        if 3<= n <= 6
            continue
        end
        println(n)
    end
```

Funciones

Definiendo una función

Una función es un objeto que toma como *input* algunos argumentos, les hace operaciones (cualquiera permitida) y regresa valores.

Nota: Pueden ser de distintos tipos y estructuras. Los argumentos pueden estar expresados por tipo (opcional pero recomendado); los tipos pueden ser definidos por uno mismo.

La sintaxis general de una función es la siguiente:

```
function funcname(argumentos)
        #Something
        return values
end
```

Ejemplo 1: Función zeta de Reimman

```
function sum_zeta(s,nterms)
    x = 0
    for n in 1:nterms
        x = x + (1/n)^s
    end
    return x
end
```

```
sum_zeta(2,10000)
```

Ejemplo 2: Varios valores de salida

```
function multi(n,m)
    n*m, div(n,n), n%m
end
```

```
multi(8,2)
```

Ejemplo: Funciones que modifican su input (!)

Regularmente los valores no son copiados cuando pasan a una función. Una forma de decirle que lo puede cambiar es con el signo de exclamación `(!)`

```
function add_one!(x)
x .= x .+ 1
end
```

```
begin
function add_one!(x)
    x .= x .+ 1
end
    x = [1,2,3]
    add_one!(x)
end
```

Funciones como expresiones matemáticas

```
f(x,y) = x^3 -x*y + 1/y;
f(4,5)
```

Funciones anónimas

- Se pueden definir funciones sin nombre

```
function (x)
    x + 2
end
```

- Funciones Lambda

```
(x) -> x + 2
```

- Funciones Lambda

```
x -> x + 2
```

Funciones de funciones

Una función puede tomar una función como argumento

```
function ff(f::Function,x::Float)
    #Operaciones con f(x)
end
```

Ejemplo: Función de función

```
function derivada(f::Function,x::Float64,dx::Float64=0.001)
        df = (f(x+dx) - f(x-dx))/(2*dx)
        return df
end

f = x-> 2*x^2 + 30*x + 9;

derivada(f,2.0,0.01)
```

Arreglos, vectores y matrices

Las matrices tienen la siguiente forma:

```
A = [1 2 3; 1 2 4; 2 2 2]
```

```
3×3 Matrix{Int64}:
 1  2  3
 1  2  4
 2  2  2
```

Es equivalente a

```
A = [1 2 3;
     1 2 4;
     2 2 2]
```

Otros arreglos se pueden escribir de la siguiente manera:

```
b1 = [4.0, 5, 6]              # 3-element Vector{Float64}
b2 = [4.0; 5; 6]              # 3-element Vector{Float64}
m1 = [4.0 5 6]                # 1×3 Matrix{Float64}
```

Arreglos `Any`

Julia puede manejar arreglos con tipos no-numéricos

```
A = ["Hello", 1, 2, 3]
  4-element Vector{Any}:
  "Hello"
  1
  2
  3
```

Arreglos indefinidos

Se puede inicializar un arreglo vacío pero con las dimensiones bien definidas

```
n = 5
A1 = Array{Float64}(undef,n,n)          # 5×5 Matrix{Float64}
A2 = Matrix{Float64}(undef,n,n)         # 5×5 Matrix{Float64}

V1 = Array{Float64}(undef,n)            # 5-element Vector{Float64}
V2 = Vector{Float64}(undef,n)           # 5-element Vector{Float64}

A = Array{String}(undef,n)
A = Array{Any}(undef,n)
```

Arreglos vacíos

Podemos inicializar arreglos indefinidos y vacíos

```
v = Array{Float64}(undef,0)

v = Array{Float64}(undef,0)

v = []    #  Any[]
```

Funciones sobre arreglos y broadcasting (Operador .)

En Julia es muy sencillo aplicar funciones a arreglos.

Ejemplo: Operador .

```
let
    f(x) = 3*x^3 / (1+x^2)
    x = [2π/n for n=1:30]
    y = f.(x)
end
```

También podemos escribirlo de la siguiente manera

```
let
    y = @. 2*x^2 + 3*x^5 - 2*x^8
end
```

Indizado y rebanado

Es fácil obtener elementos de arreglos a través del indizado. Julia está muy optimizado para hacer ese tipo de labores.

Ejemplo: Primer y último elemento

```
let
    A = rand(6)
    b = A[begin]
    e = A[end]
    println(A)
    println()
    println("primer elemento es $b"," y ","último elemento es = $e")
end
```

Ejemplo: Rebanado

```
let
    A = rand(6,6)
    display(A)
    B = A[begin:2:end,begin:2:end]
    display(B)
    C = A[1:2:5,1:2:5]
    display(C)
end
```

Ejemplo: Indizado lógico

```
let
    A =  rand(6,6)
    display(A)
    A[A .< 0.5] .= 0
    display(A)

end
```

Ejemplo: Iteraciones sobre arreglos

```
let
    A = rand(6)
        for i ∈ eachindex(A)
            println(string("i=$(i) A[i]=$(A[i])"))
        end
end
```

Otros comandos útiles

- `firstindex(A,dim)`
- `lastindex(A,dim)`
- `similar(Array{Float64}, axes(A))`

Operaciones con arreglos

Ejemplo: Multiplicación

```
let
    A = rand(3,3)
    display(A)
    B = rand(3,3)
    display(B)
    C = A*B
    display(C)
end
```

```
let
    A = rand(4,4)
    display(A)
    v = rand(4)
    display(v)
    w = A*v
    display(w)
end
```

Ejemplo: Multiplicación elemento por elemento

```
let
    A = rand(3,3)
    display(A)
    B = rand(3,3)
    display(B)
    C = A.*B
    display(C)
end
```

Ejemplo: Producto punto

```
let
    v = rand(100)
    w = rand(100)
    z = dot(v,w)
end
```

Ejemplo: Operador backslash

```
let
    A = rand(3,3)
    b1 = [4.0, 5, 6]              # 3-element Vector{Float64}
    b2 = [4.0; 5; 6]              # 3-element Vector{Float64}
    m1 = [4.0 5 6]                # 1×3 Matrix{Float64}

    x=A\b1                        # Solves A*x=b
    x=A\b2                        # Solves A*x=b
    #x=A\m1                       # Error!!
end
```

Rearreglo y concatenación

Rearreglo en una dimensión

Podemos usar `push!` y `pop!`

Ejemplo

```
let
A = Float64[]        # Equivalent to A=Array{Float64}(undef,0)
push!(A, 4)          # Adds the number 4 at the end of the array
push!(A, 3)          # Adds the number 3 at the end of the array
v = pop!(A)          # Returns 3 and removes it from A
end
```

Concatenación

Podemos usar los comandos `hcat`, `vcat`, `cat`

Ejemplo: `vcat`, `hcat`

```
md"""###### Ejemplo: `vcat`, `hcat` """
```

```
let
    A = [4 5 6]
    B = [6 7 8]

    M1 = vcat(A,B)
    display(M1)

    M2 = hcat(A,B)
    display(M2)
end
```

Ejemplo: cat

La función `cat` es más general y permite realizar operaciones para cualquier dimensión

```
let
    A = [4 5 6]
    B = [6 7 8]
    M1 = cat(A,B, dims=1)
    display(M1)

    M2 = cat(A,B, dims=2)
    display(M2)

    M3 = cat(A,B, dims=3)
    display(M3)
end
```

Concatenación de matrices

Ejemplo

```
let
    A = [4 5 6]
    B = [6 7 8]

    M1 = [A;B]
    display(M1)

    M2 = [A B]
    display(M2)

end
```

Estructura de datos

Algunas de las estructuras más usadas son: tuples, named tuples y dictionaries.

Inicializando tuplas

Ejemplo:

```
let
    t = (3.14,2.8) # Tuple{Float64,Float64}
    display(t)
end
```

```
let
    t = 3.24, 2.8
    display(t)
end
```

Convertir tuplas a arreglos

Ejemplo

```
let
    a = (1,2,3)
    t1 = collect(a)
    display(t1)

    t2 = [x for x in a]
    display(t2)

    t3 = [a...]
    display(t3)
end
```

Named tuples

Ejemplo

```
let
    p = (x = 1.1, y = 2.4) #NamedTuple{(:x, :y, Tuple{Float64, Float64}}

    K = keys(p)
    display(K)
    V = values(p)
    display(V)
end
```

Diccionarios

Un diccionario es una colección de pares de key-values

Creando un diccionario

Ejemplo

```
let
    D1 = Dict("a" => 1, "b"=> 2, 1 => "a")
    display(D1)

    D2 = Dict([("a",1),("b",2),(1,"a")])
    display(D2)
end
```

Accediendo a los elementos

Ejemplo

```
let
    D = Dict([("a", 1), ("b", 2), (1,"a")])
    #D["a"]
    D[1]
end
```

```
let
    D = Dict([("a", 1), ("b", 2), (1,"a")])
    for e in D
        println(e)
    end
end
```

```
let
    D = Dict([("a", 1), ("b", 2), (1,"a")])
    for (k,v) in D
        println(k,"=>",v)
    end
end
```

Modificando un diccionario

Ejemplo

```
let
    D = Dict([("a", 1), ("b", 2), (1,"a")])
    D["c"] = 3
    display(D)
    D["c"] = "Hola"
    display(D)
    D = delete!(D,"c")
    display(D)
end
```

Estructuras

Las estructuras definen un nuevo tipo coompuesto, están basadas en campos dados y tienen diferentes tipos. Una vez inicializadas no hay forma de modificarlas.

Ejemplo

```
struct Location
name::String
lat::Float64
lon::Float64
end
```

```
let
    struct Location
    name::String
    lat::Float64
    lon::Float64
    end

    loc1 = Location("Los Angeles", 34.0522,-118.2437)

    display(loc1.name)
    display(loc1.lat)

end
```

Ejemplo: Difinir vector con una estructura

```
let
    sites = Location[]
    push!(sites,Location("Los Angeles", 34.0522,-118.2437))
    push!(sites,Location("Las Vegas", 36.1699,-115.1398))

    display(sites)
end
```

Estructuras mutables

Si queremos modificar las componentes de una estructura después de haber sido inicializada, debemos usar una estructrua mutable.

```
mutable struct mLocation
    name::String
    lat::Float32
    lon::Float32
end
```

Ejemplo

```
let
    mutable struct mLocation
        name::String
        lat::Float32
        lon::Float32
    end
end
```

```
let
loc1 = mLocation("Los Angeles", 34.0522,-118.2437)
loc1.name = "LA"
end
```

```
Enter cell code...
```

Gráficas

Existen varios paquetes para graficar

- Plots
- Plotly
- Pyplot
- Makie
- ...

```
using Plots
```

Ejemplos

```
let
x = 0:0.05:1;
y = sin.(2π*x);
plot(x,y)
end
```

```
let
x = 0:0.05:1;
y = sin.(2π*x);
plot(x,y,
    seriestype =:path,
    linestyle = :dash
)
end
```

```
let
    x = 0:0.05:1;
y = sin.(2π*x);

plot(   x, y,
        seriestype=:sticks,
        linestyle=:dash,
        lw = 3,
        seriescolor = :green,
        marker = :circle,
        markersize = 8,
        markercolor = :green,
        markerstrokecolor = :green,
    )
end
```

```
let
x = 0:0.05:1;
y = sin.(2π*x);

plot(   x, y,
        title="title (titlefontsize)",
        xlabel="xlabel (guidefontsize)",
        ylabel="ylabel (guidefontsize)",
        label="sin(2*pi*x) (legendfontsize)",
        titlefontsize=18,
        guidefontsize=18,
        tickfontsize=16,
        legendfontsize=12,
        legend=:topright
    )
end
```

```
using LaTeXStrings
```

```
let
    x = 0:0.05:1;
y = sin.(2π*x);
plot(   x, y,
        line=(3,:green,:dash,:sticks),
        marker=(:circle,8,:green,:green),
        title="title",
        xlabel="xlabel",
        ylabel="ylabel",
        label=L"\sin(2\pi x)",
        legend=:outertopright,
        titlefontsize=18,
        guidefontsize=18,
        tickfontsize=16,
        legendfontsize=18,
        grid=false
    )
end
```

Algunas de las opciones para la posición de `legend` son:

- :right
- :left
- :top
- :bottom
- :topright
- :topleft
- :bottomright
- :bottomleft
- :outerright
- :outerleft
- :outertopright
- :outertopleft
- :outerbottomright
- :outerbottomleft

```
let
x = 0:0.05:1;
y = sin.(2π*x);
p1 = scatter(x, y, label="(x,y)")
p2 = scatter(x, -y, label="(x,-y)", legend=:topleft)
p3 = scatter(-x, y, label="(-x,y)", legend=:topleft)
p4 = scatter(-x, -y, label="(-x,-y)")

plot(p1,p2,p3,p4, layout = (2,2) )
end
```

```
let
x = 0:0.01:1;
y = @. sin(8*2π*x)/exp(10*x);

plot(
    x,y,
    lw=3,
    label=false,
    ylimits=(-1,1),
    xlimits=(0,0.5)
    )
end
```

```
let
    f(x) = exp(x)
p1 = plot(f, 1, 10, lw=3,
            title="Regular Plot of exp(x)",
            label=false)

p2 = plot(f, 1, 10,
            yscale=:log10, lw=3,
            title="Semilog-y plot of exp(x)",
            label=false)

plot(p1,p2,layout=(2,1))
end
```