Programación en Julia: Herramientas para el aprendizaje

Notas interactivas en Pluto

Héctor Medel

Benjamín Pérez

¿Qué es Pluto?

Es un ambiente de programación para Julia de tipo *notebook* (con soporte para markdown) que tiene las siguientes características:

- Reactividad
- Interactividad
- Simplicidad

Es útil para educación y exploración en proyectos más ambiciosos.

Generalidades del entorno

- Los *shortcuts* del teclado podemos obtenerlos al presionar F1.
- Barra de exportación.
- Cuadro de texto para grabar.
- Documentación.
- Navegación en celdas

Reactividad

¿A qué nos referimos con reactividad?

Pluto rastrea las dependencias entre celdas, y actualiza los valores de acuerdo a dicha cadena de dependencias.

```
a = 1
```

```
b = 10
```

```
c = 2.1
```

```
c = a/b + 2
```

Algunos puntos importantes:

- La reactividad nos permite hacer un acomodo *no usual* de las celdas.
- En cada celda solo hemos puesto una instrucción (hasta ahora).
- El rastreo de las dependencias de las celdas, crea un problema si volvemos a definir variables en celdas posteriores.

```
a = 0
```

Bloques de código

Para ejecutar varias líneas de código desde una celda tenemos dos opciones:

```
begin
    CODIGO
    CODIGO
    CODIGO
end
```

```
let
    CODIGO
    CODIGO
    CODIGO
end
```

¿Qué diferencia hay?

5

```
begin # El scope de las variables definidas es global
    d = 2
    e = 2+a+d
end
```

5

```
let # El scope de las variables definidas es local
    f = 2
    e = 2+a+f
end
```

```
f
```

2

```
d
```

Interactividad

Una manera de agregar elementos interactivos (sliders, botones, etc) es usando el paquete PlutoUI.jl. Para esto, ejecutaremos el siguiente comando en una nueva celda

```
using PlutoUI
```

```
using PlutoUI ✓
```

Sliders

Tenemos la siguiente sintaxis:

```
@bind VARIABLE Slider(RANGO, VALOR POR DEFECTO, TRUE/FALSE)
```

5

```
@bind sliderA Slider(0:10, 5, true)
```

15

Para múltiples sliders usaremos el símbolo $ dentro del entorno de md.

B 5

C 5

```
md"
B $(@bind sliderB Slider(0:10, 5, true))

C $(@bind sliderC Slider(0:10, 5, true))
"
```

10

```
sliderB+sliderC
```

Scrubable

La sintaxis es muy similar a la de Slider. Veamos.

```
@bind VARIABLE Scrubbable(RANGO, default=VALOR POR DEFECTO)
```

5

```
@bind scrubD Scrubbable(0:10, default=5)
```

Number field

La sintaxis es la siguiente:

```
@bind VARIABLE NumberField(RANGO, VALOR POR DEFECTO)
```

1

```
@bind numeroE NumberField(-5:5, default=1)
```

2

```
2*numeroE
```

TextField

La sintaxis es:

```
@bind VARIABLE TextField(default = VALOR POR DEFECTO)
```

```
@bind textF TextField()
```

""

```
textF
```

```
@bind numeroG TextField()
```

```
""
```

```
numeroG # Esta es una variable tipo char
```

```
ArgumentError: cannot parse "" as Float64

  1. _parse_failure(::Type, ::String, ::Int64, ::Int64) @ parse.jl:373
  2. #tryparse_internal#478 @ parse.jl:369 [inlined]
  3. tryparse_internal @ parse.jl:366 [inlined]
  4. #parse#479 @ parse.jl:379 [inlined]
  5. parse(::Type{Float64}, ::String) @ parse.jl:379
  6. top-level scope @ Local: 1 [inlined]
```

```
parse(Float64, numeroG)
```

CheckBox

La sintaxis es:

```
@bind checkG CheckBox()
```

Palomeamos el cuadro ☐

```
md"
Palomeamos el cuadro $(@bind checkG CheckBox())
"
```

```
false
```

```
checkG
```

Button

La sintaxis es:

```
@bind buttonH Button("TEXTO")
```

Botón

```
@bind buttonH Button("Botón")
```

```
"Botón"
```

```
buttonH
```

```
0.5841756323870744
```

```
begin
    buttonH
    rand()
end
```

Clock

La sintaxis es la siguiente:

```
@bind t Clock()
```

Start speed: 1 *secs / tick*

```
@bind t Clock()
```

```
deltaT = 0.1
```

```
deltaT = 1/10
```

Start

```
@bind tf Clock(deltaT, true)
```

5266

```
tf
```

Gráficas

Queremos generar gráficas que sean manipulables vía sliders, cuadros de texto u otras entradas interactivas.

Vamos a explorar la función

$$f(x) = Ae^{-(x-x_0)^2/w^2}$$

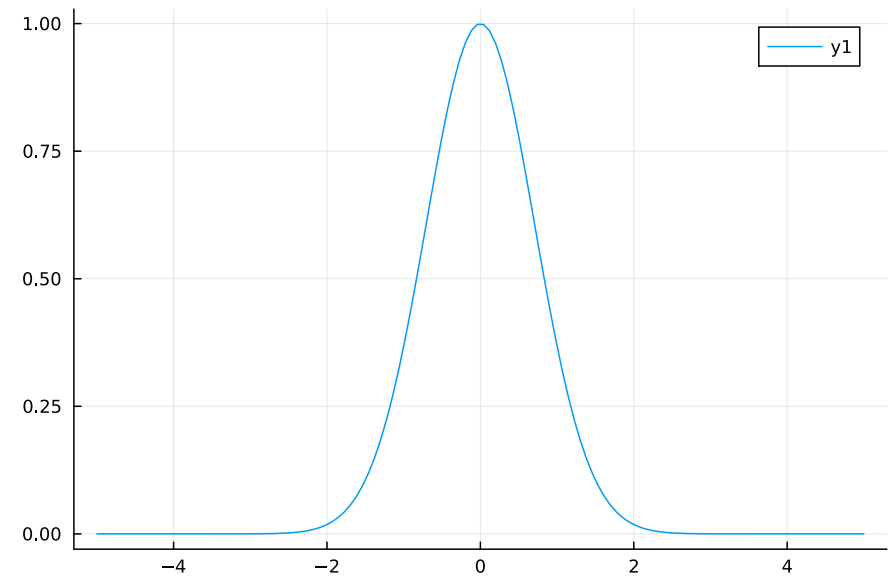
```
using Plots
```

```
f (generic function with 1 method)
```

```
f(x; A=1.0, w=1.0) = A * exp(-x^2/w^2)
```

```
(1.0, 0.735759, 1.0)
```

```
f(0), f(1; A=2), f(0; w=2)
```



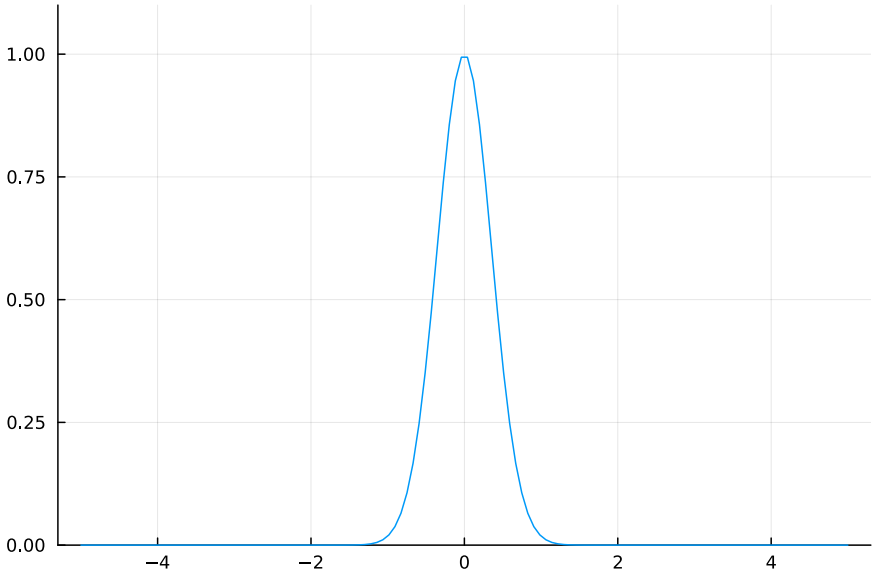

```
begin
    xs = range(-5.0, 5.0, 128)
    plot(xs, f.(xs))
end
```

Vamos a generar los sliders para manipular la gráfica.

xo 0.0

Ao 1.0

wo 0.5



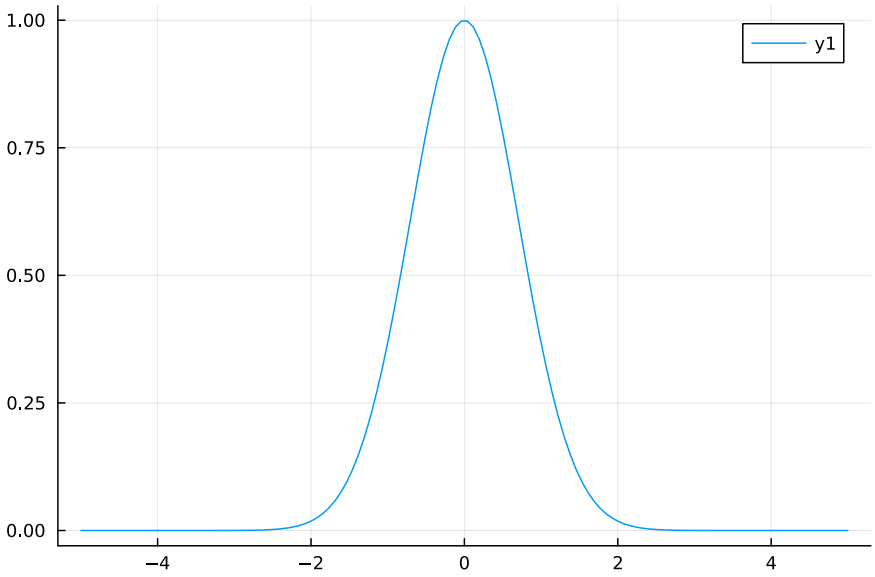

```
begin
    # xs = range(-5.0, 5.0, 128) # Ya esta definida en la celda anterior!
    ys = f.(xs.-x0; A=A0, w=w0)
    plot(xs, ys, ylims=(0,1.1), leg=:false)
end
```

```
g (generic function with 1 method)
```

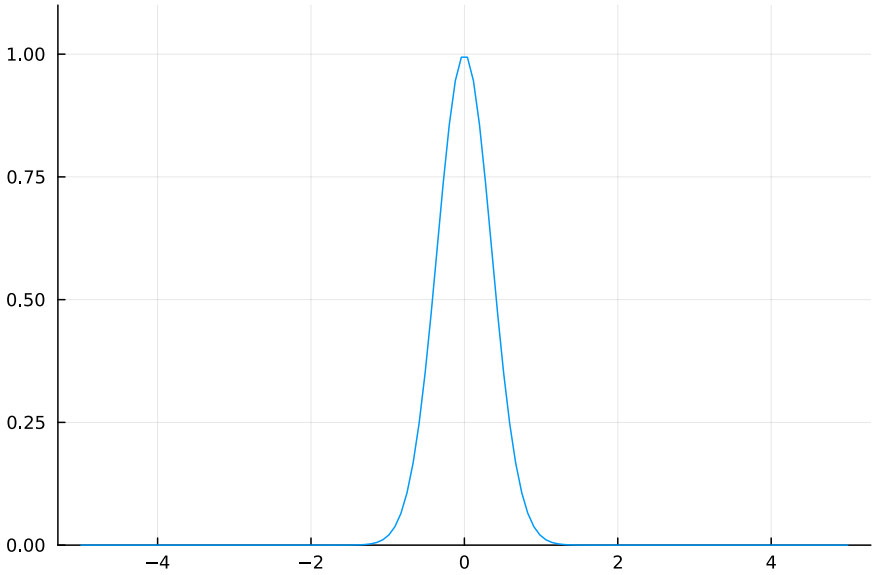
```
g(x, A, w) = A * exp(-x^2/w^2)
```

```
(1.0, 0.735759, 1.0)
```

```
g(0,1,1), g(1, 2,1), g(0,1,1)
```




```
begin
    # xs = range(-5.0, 5.0, 128)
    plot(xs, g.(xs,1,1))
end
```




```
begin
    # xs = range(-5.0, 5.0, 128)
    zs = g.(xs.-x0,A0,w0)
    plot(xs, zs, ylims=(0,1.1), leg=:false)
end
```

Otros ajustes

Tabla de contenidos

PlutoUI permite agregar una tabla de contenidos de acuerdo a los *headlines* del notebook.

Para agregar ejecutamos el siguiente comando en una celda

```
TableOfContents(title="Contenidos", indent=true/false, depth=3 ,aside=true/false)
```

```
#TableOfContents(title="Contenidos", indent=true, aside=true)
```

Ancho de las celdas

Es posible agregar código en html o css. Esto nos puede ayudar a hacer ajustes finos de las notas que estamos generando.

Para cambiar el ancho de las celdas, usaremos la celda en modo html. Ejecutaremos la siguiente instrucción en una celda:

```
html" <style>
main {
    max-width: ANCHOpx;
}
"
```

```
html"<style>
main {
    max-width: NNpx;
}
"
```