Julia Onramp

A. Mhamdi

January 3, 2024

1 GOALS

- Enter commands in `Julia` REPL to create variables and perform calculations;
- Write and save programs;
- Use indexing to extract and modify rows, columns, and elements of `Julia` tensors.

`Julia` is a standalone program which can be downloaded from https://julialang.org/downloads/

Getting around

By default, Julia runs in an interactive terminal called the REPL. In this mode, Some useful commands are: 1. ^C aborts execution 1. ^D exits Julia 1. ? enters help mode 1. ; enters system shell mode 1.] enters package manager mode 1. `^l` clears screen

We begin first by activating the environement within the desired folder.

From the REPL interface, either type

```
using Pkg
pkg"activate ."
```

or access the package mode by typing] and simply write

```
activate .
```

Always within the package mode, to see the full list of installed packages

```
st
```

[1]:
```
using Pkg
pkg"activate ."
```

```
  Activating project at `~/MEGA/git-repos/infodev/Codes`
```

[2]:
```
]st
```

```
Status `~/MEGA/git-repos/infodev/Codes/Project.toml`
  [336ed68f] CSV v0.10.11
  [a93c6f00] DataFrames v1.6.1
  [7073ff75] IJulia v1.24.2
⌅ [ee78f7c6] Makie v0.19.12
  [5deeb4b9] Mousetrap v0.3.1
`https://github.com/clemapfel/mousetrap.jl#main`
```

```
  [5fb14364] OhMyREPL v0.5.23
  [91a5bcdd] Plots v1.39.0
  [c3e4b0f8] Pluto v0.19.32
  [d6f4376e] Markdown
Info Packages marked with ⌃ have new versions
available and may be upgradable.
```

To add the **Markdown** package for instance, we write

[3]:
```
]add Markdown
```

```
   Resolving package versions...
  No Changes to `~/MEGA/git-repos/infodev/Codes/Project.toml`
  No Changes to `~/MEGA/git-
repos/infodev/Codes/Manifest.toml`
```

To be able to use it, we do as follows

[4]:
```
using Markdown
```

[5]:
```
md"""
This a text inside a code cell, thanks to **Markdown** package. I⌃
  ↪can _emphasize_ anything. Make other things **bold**
"""
```

[5]:

This a text inside a code cell, thanks to **Markdown** package. I can *emphasize* anything. Make other things **bold**

Runnig `Julia` in Jupyer Notebookor Jupyter Lab is pretty handy. We only need to install the appropriate kernel. In order to add `Julia` kernel `IJulia` to Jupyter Notebook and/or `JupyterLab` IDEs, we begin by executing the following commands:

```
using Pkg
Pkg.add("IJulia")
```

If we want to get `JupyterLab` instance running in current directory, we can do:

```
jupyterlab(dir=pwd(), detached=true)
```

In case things do not work, we run the two following commands from `Julia` REPL which launch jupyter environment.

```
using IJulia
installkernel("Julia")
```

The shell mode is also available through the REPL to evaluate some os commands. To do so, simply preface the regular command by semicolon. For instance, pwd prints the path to working directory and `ls` allows to list the content of the current directory.

[6]:
```
;pwd
```

```
/home/mhamdi/MEGA/git-repos/infodev/Codes
```

[7]:
```
;ls -la
```

```
total 752
drwxrwxr-x 3 mhamdi mhamdi   4096 Nov 29 21:29 .
drwxrwxr-x 6 mhamdi mhamdi   4096 Nov 19 16:27 ..
drwxrwxr-x 2 mhamdi mhamdi   4096 Nov 29 21:09 .ipynb_checkpoints
-rw------- 3 mhamdi mhamdi 617224 Nov 29 21:29 julia-onramp.ipynb
-rw------- 1 mhamdi mhamdi  34286 Jan 14  2023 Julia.png
-rw-rw-r-- 1 mhamdi mhamdi  83778 Nov 29 21:13 Manifest.toml
-rw-rw-r-- 1 mhamdi mhamdi    444 Nov 29 21:13 Project.toml
-rw-rw-r-- 1 mhamdi mhamdi     20 Nov 19 16:27 README.md
-rw-rw-r-- 1 mhamdi mhamdi     66 Nov 29 21:28 test-file.csv
-rw------- 1 mhamdi mhamdi     18 Nov 29 21:17 .wakatime-project
```

2 Getting Help

In order to seek help on a particular function. We just use the ? mark. We can use the Julia documentation to discover more pieces of information about Julia features.

[8]:
```
?cos
```

```
search: cos
cosh
cosd
cosc
cospi
acos
acosh
acosd
sincos
sincosd
sincospi
const
```

[8]:

```
cos(x)
```

Compute cosine of x, where x is in radians.

See also cosd, cospi, sincos, cis.

```
cos(A::AbstractMatrix)
```

Compute the matrix cosine of a square matrix A.

If A is symmetric or Hermitian, its eigendecomposition (eigen) is used to compute the cosine. Otherwise, the cosine is determined by calling exp.

3 Examples

```
julia> cos(fill(1.0, (2,2)))
2×2 Matrix{Float64}:
  0.291927  -0.708073
 -0.708073   0.291927
```

To print something on the standard output, it is possible to use either `print` and `println`. The last one displays the text and moves the cursor to the next line.

[9]:
```
print("Hello")
print(' ')
print("World")
```

```
Hello World
```

[10]:
```
println("Hello")
println("World")
```

```
Hello
World
```

Data types: Dictionaries

[11]:
```
dict = Dict(
    # Name => # of wheels
    "Unicycle" => 1,
    "Bicycle" => 2,
    "Tricycle" => 3
)
```

[11]:
```
Dict{String, Int64} with 3 entries:
  "Bicycle"  => 2
  "Tricycle" => 3
  "Unicycle" => 1
```

[12]:
```
typeof(dict)
```

[12]:
```
Dict{String, Int64}
```

[13]:
```
dict = Dict([("Unicycle", 1), ("Bicycle", 2), ("Tricycle", 3)])
```

[13]:
```
Dict{String, Int64} with 3 entries:
  "Bicycle"  => 2
  "Tricycle" => 3
  "Unicycle" => 1
```

[14]:
```
dict["Bicycle"]
```

[14]:
```
2
```

[15]:
```
lst = [1, 'a', "abc", true, [0, .5im]]
```

[15]:
```
5-element Vector{Any}:
     1
      'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)
      "abc"
 true
      ComplexF64[0.0 + 0.0im, 0.0 + 0.5im]
```

[16]:
```
typeof(lst)
```

[16]:
```
Vector{Any} (alias for Array{Any, 1})
```

[17]:
```
lst[end]
```

[17]:
```
2-element Vector{ComplexF64}:
 0.0 + 0.0im
 0.0 + 0.5im
```

3.0.1 Basic Calculations

[18]:
```
a, b = 1, 1.5
```

[18]:
```
(1, 1.5)
```

[19]:
```
println(typeof(a))
println(typeof(b))
```

```
Int64
Float64
```

[20]:
```
md"""
`varinfo` method allows to display loaded variables.
"""
```

[20]:
`varinfo` method allows to display loaded variables.

[21]:
```
varinfo()
```

[21]:

name	size	summary
Base		Module
Core		Module
Main		Module
PLOTS_DEFAULTS	456 bytes	Dict{Symbol, Symbol} with 1 entry
a	8 bytes	Int64
b	8 bytes	Float64
dict	503 bytes	Dict{String, Int64} with 3 entries
lst	176 bytes	5-element Vector{Any}
showall	0 bytes	showall (generic function with 1 method)

[22]:
```
?varinfo
```

```
search: varin
0mfo
```

[22]:

`varinfo(m::Module=Main, pattern::Regex=r""; all::Bool = false, imported::Bool =`

Return a markdown table giving information about exported global variables in a module, optionally restricted to those matching `pattern`.

The memory consumption estimate is an approximate lower bound on the size of the internal structure of the object.

- `all` : also list non-exported objects defined in the module, deprecated objects, and compiler-generated objects.
- `imported` : also list objects explicitly imported from other modules.
- `recursive` : recursively include objects in sub-modules, observing the same settings in each.
- `sortby` : the column to sort results by. Options are `:name` (default), `:size`, and `:summary`.
- `minsize` : only includes objects with size at least `minsize` bytes. Defaults to 0.

[23]:
```
println("Sum of $a and $b is $(a+b)")
```

```
Sum of 1 and 1.5 is 2.5
```

Addition, subtraction, multiplication, division, exponent

[24]:
```
a+b, a-b, a*b, a÷b, a^b
```

[24]:
```
(2.5, -0.5, 1.5, 0.0, 1.0)
```

[25]:
```
md"""
**Unicode support**
1. We can use α instead of `pi`
1. Greek letters improe comprehension: α _(alpha)_, β _(beta)_, ...
1. Symbols: ≥ _(>=)_, ≤ _(<=)_, ∈ _(in)_, ...
1. ...
```

```
"""
```

[25]:

Unicode support

1. We can use π instead of `pi`
2. Greek letters improe comprehension: α *(alpha)*, β *(beta)*, ...
3. Symbols: ≥ *(>=)*, ≤ *(<=)*, ∈ *(in)*, ...
4. ...

[26]:
```
3 □ □
```

[26]: true

[27]:
```
typeof(3.14)
```

[27]: Float64

[28]:
```
Float64 |> supertype |> supertype |> supertype |> supertype
```

[28]: Any

[29]:
```
Integer |> subtypes
```

[29]:
```
3-element Vector{Any}:
 Bool
 Signed
 Unsigned
```

[30]:
```
Signed |> subtypes
```

[30]:
```
6-element Vector{Any}:
 BigInt
 Int128
 Int16
 Int32
 Int64
 Int8
```

[31]:
```
UInt8 <: Unsigned # UInt8 is one subtype of Unsigned
```

[31]: true

[32]:
```
Signed >: Int8 # Signed is supertype of Int8
```

[32]: true

[33]:
```
typeof(3)
```

[33]:
```
Int64
```

[34]:
```
tmp::UInt8 = 3
typeof(tmp)
```

[34]:
```
UInt8
```

3.0.2 Mathematical Notation

[35]:
```
println(1+2)
println(+(1, 2))
```

```
3
3
```

[36]:
```
println(1-2)
println(-(1, 2))
```

```
-1
-1
```

[37]:
```
println(1*2)
println(*(1, 2))
```

```
2
2
```

[38]:
```
println(1/2)
println(/(1, 2))
```

```
0.5
0.5
```

[39]:
```
println(3/4+7/5)
println(3//4+7//5)
```

```
2.15
43//20
```

3.0.3 Array Transformations

Perform calculations on entire arrays at once.

[40]:
```
zeros(3, 2)
```

[40]:
```
3×2 Matrix{Float64}:
 0.0  0.0
 0.0  0.0
 0.0  0.0
```

[41]:
```
ones(3, 3, 2)
```

[41]:
```
3×3×2 Array{Float64, 3}:
[:, :, 1] =
 1.0  1.0  1.0
 1.0  1.0  1.0
 1.0  1.0  1.0

[:, :, 2] =
 1.0  1.0  1.0
 1.0  1.0  1.0
 1.0  1.0  1.0
```

[42]:
```
fill(□, (2, 2))
```

[42]:
```
2×2 Matrix{Irrational{:□}}:
 □  □
 □  □
```

[43]:
```
□ .* ones(2, 2)
```

[43]:
```
2×2 Matrix{Float64}:
 3.14159  3.14159
 3.14159  3.14159
```

[44]:
```
md"Creates a `BitArray` with all values set to `true`"
```

[44]:

Creates a `BitArray` with all values set to `true`

[45]:
```
var = trues(2, 4)
println(var)
typeof(var)
```

```
Bool[1 1 1 1; 1 1 1 1]
```

[45]:
```
BitMatrix (alias for BitArray{2})
```

[46]:
```
md"Creates a `BitArray` with all values set to `false`"
```

[46]:

Creates a `BitArray` with all values set to `false`

[47]:
```
var = falses(2, 4)
println(var)
```

```
typeof(var)
```

```
Bool[0 0 0 0; 0 0 0 0]
```

[47]:
```
BitMatrix (alias for BitArray{2})
```

[48]:
```
md"**Comprehension**"
```

[48]:

Comprehension

[49]:
```
str = "Hello Julia"
[println(el) for el in str];
```

```
H
e
l
l
o

J
u
l
i
a
```

3.0.4 Calling Functions

Call functions to obtain multiple outputs.

[50]:
```
md"""
[Functions in ${\tt Julia}$](https://docs.julialang.org/en/v1/
  ↪manual/functions/)
"""
```

[50]:

Functions in `Julia`

[51]:
```
md"**Spreading Arguments**"
```

[51]:

Spreading Arguments

Optional positional arguments

[52]:
```
foo(x=0, y=0, z=0) = x+y+z
```

```
foo (generic function with 4 methods)
```

[53]:
```
foo(), foo(1, 2, 3)
```

```
(0, 6)
```

[54]:
```
foo([1, 2, 3]...) # Splat `...` operator
```

[54]: 6

Keywords arguments

[55]:
```
bar(; a::Real=0, b::Real=0, c::Real=0) = a+b+c
```

[55]: bar (generic function with 1 method)

[56]:
```
bar()
```

[56]: 0

[57]:
```
bar(; Dict(:a => 3, :b => 5.4, :c => -1.2)... ) # ; kwargs...
```

[57]: 7.2

[58]:
```
# THROW AN ERROR
try bar([1, 2, 3]...)
catch error
    println(error)
end
```

MethodError(bar, (1, 2, 3), 0x00000000000082db)

[59]:
```
md"**Multiple Dispatch**"
```

[59]:

Multiple Dispatch

[60]:
```
# 1st method signature
function f(x::Int)
    x^2
end
```

[60]: f (generic function with 1 method)

[61]:
```
# 2nd method signature
f(x::Float64) = x^2+1
```

[61]: f (generic function with 2 methods)

[62]:
```
# 3rd method signature
f(x::Char) = x*'y'*'z'
# 4th mehod signature
f(x::String) = x*x
```

[62]: f (generic function with 4 methods)

```
[63]: methods(f)
```

```
[63]: # 4 methods for generic function "f" from Main:
       [1] f(x::Int64)
           @ In[60]:2
       [2] f(x::Float64)
           @ In[61]:2
       [3] f(x::Char)
           @ In[62]:2
       [4] f(x::String)
           @ In[62]:4
```

```
[64]: f(1), f(1.), f('x'), f("abc")
```

```
[64]: (1, 2.0, "xyz", "abcabc")
```

```
[65]: mycos(x) = cos(x)
      mycos(adj, hyp) = adj/hyp # Extension to `mycos` function
```

```
[65]: mycos (generic function with 2 methods)
```

```
[66]: methods(mycos)
```

```
[66]: # 2 methods for generic function "mycos" from Main:
       [1] mycos(x)
           @ In[65]:1
       [2] mycos(adj, hyp)
           @ In[65]:2
```

```
[67]: @which mycos(1)
```

```
[67]: mycos(x)
           @ Main In[65]:1
```

```
[68]: @which mycos(5, 3)
```

```
[68]: mycos(adj, hyp)
           @ Main In[65]:2
```

```
[69]: mycos(adj, hyp=10) = adj/hyp
```

```
[69]: mycos (generic function with 2 methods)
```

```
[70]: @which mycos(1)
```

```
[70]: mycos(adj)
           @ Main In[69]:1
```

Function Chaining applies a function to the preceding argument.

[71]:
```
g(x) = x+1
h(x) = x^2
x = 2 |> g |> h
```

[71]: 9

[72]:
```
md"Another pssible way is t use `∘`_\circ{tab}_ symbol"
```

[72]: Another pssible way is t use ∘ *\circ{tab}* symbol

[73]:
```
(h∘g)(2)
```

[73]: 9

[74]:
```
md"Definition of a function can be done on the fly"
```

[74]: Definition of a function can be done on the fly

[75]:
```
y = 5 |> (x->x^2) |> √
```

[75]: 5.0

[76]:
```
md"""
**Metaprogramming:** Code is optimized by nature in ${\tt Julia}$
"""
```

[76]: **Metaprogramming:** Code is optimized by nature in `Julia`

[77]:
```
function Foo(x::Integer)
    y = x
    for i=1:100
        y += i^2
    end
    return y
end
```

[77]: Foo (generic function with 1 method)

[78]:
```
@code_llvm Foo(3)
```

```
;  @ In[77]:1 within `Foo`
define i64 @julia_Foo_4232(i64
signext %0) #0 {
top:
;  @ In[77]:3 within `Foo`
  %1 = add i64 %0, 338350
;  @ In[77]:6 within `Foo`
```

```
  ret i64 %1
}
```

[79]:
```
?@code_llvm
```

[79]:
```
@code_llvm
```

Evaluates the arguments to the function or macro call, determines their types, and calls `code_llvm` on the resulting expression. Set the optional keyword arguments `raw`, `dump_module`, `debuginfo`, `optimize` by putting them and their value before the function call, like this:

```
@code_llvm raw=true dump_module=true debuginfo=:default f(x)
@code_llvm optimize=false f(x)
```

`optimize` controls whether additional optimizations, such as inlining, are also applied. `raw` makes all metadata and dbg.* calls visible. `debuginfo` may be one of `:source` (default) or `:none`, to specify the verbosity of code comments. `dump_module` prints the entire module that encapsulates the function.

3.0.5 Plotting Data

Visualize variables using `Julia`'s plotting functions.

[80]:
```
]add Plots
```

```
  Resolving package versions...
  No Changes to `~/MEGA/git-repos/infodev/Codes/Project.toml`
  No Changes to `~/MEGA/git-
repos/infodev/Codes/Manifest.toml`
```

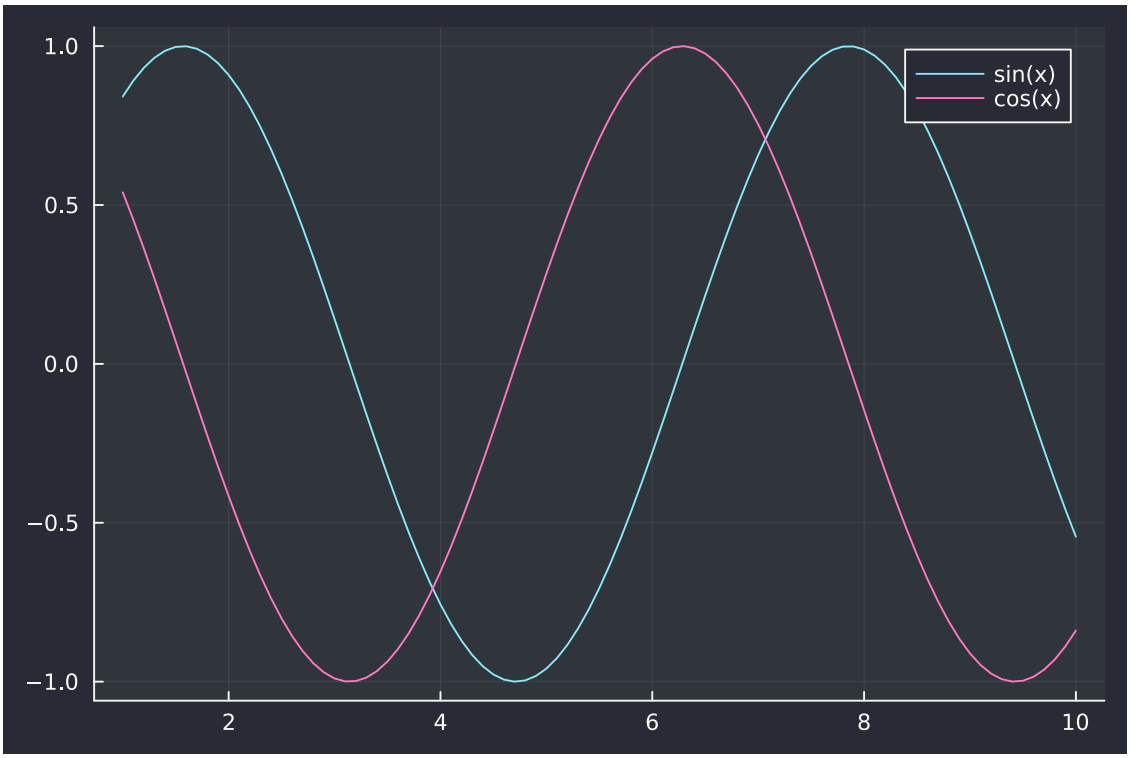
[81]:
```
using Plots # GR is the default backend
```

```
WARNING: using Plots.bar in module Main conflicts with an existing
 ↪identifier.
```

[82]:
```
x = 1:.1:10
y = sin.(x)
z = cos.(x)

plot(x, y, label="sin(x)")
plot!(x, z, label="cos(x)") # Hold on the previous plot
```

[82]:



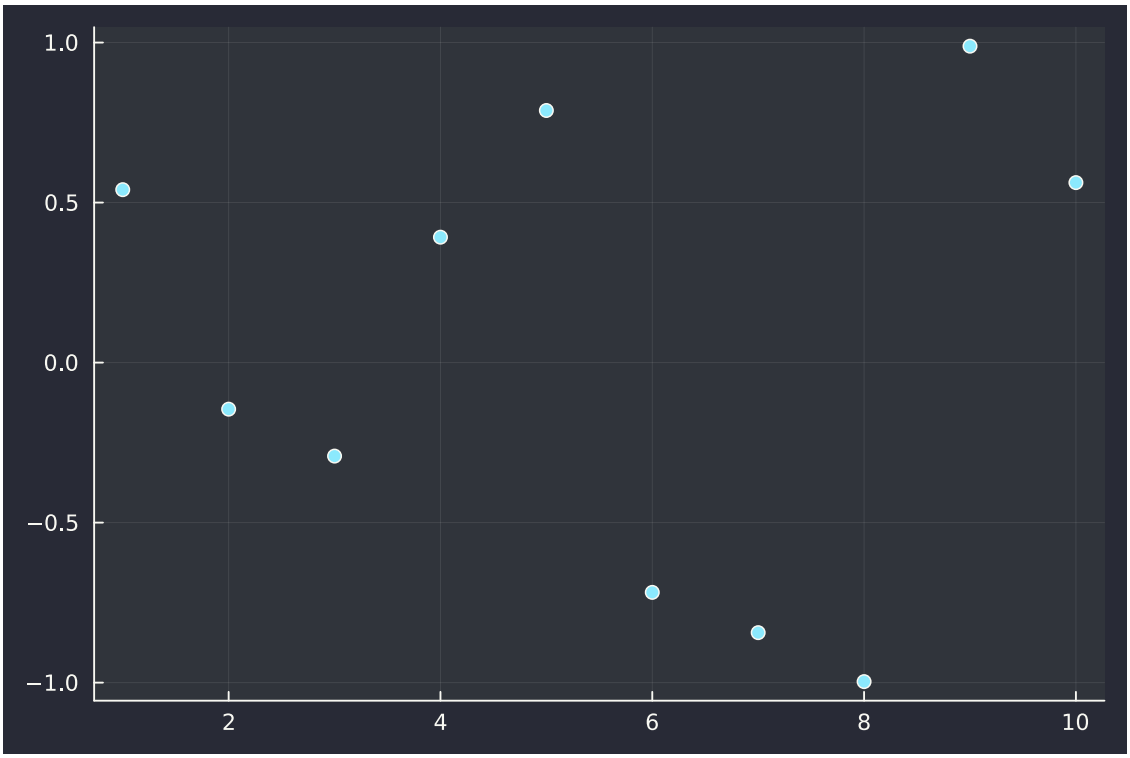
[83]:
```
md"**Scatter Plot**"
```

[83]:

Scatter Plot

[84]:
```
x = range(1, 10)
y = cos.(x.^3)
scatter(x, y, legend=false)
```

[84]:



[85]:
```
md"**Uniform Distribtion**"
```

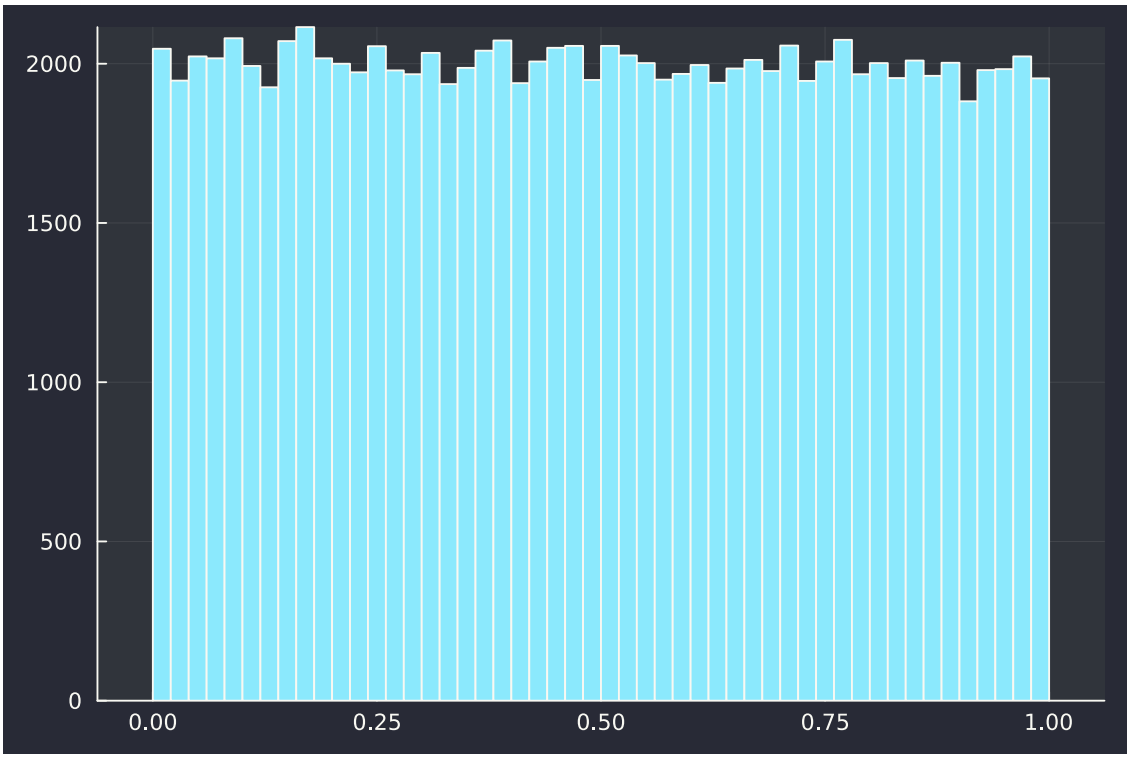
[85]:

Uniform Distribtion

[86]:
```
?rand;
```

[87]:
```
x = rand(10^5)
histogram(x, bins=64, legend=false)
```

[87]:



[88]:
```
md"**Normal Distribution**"
```

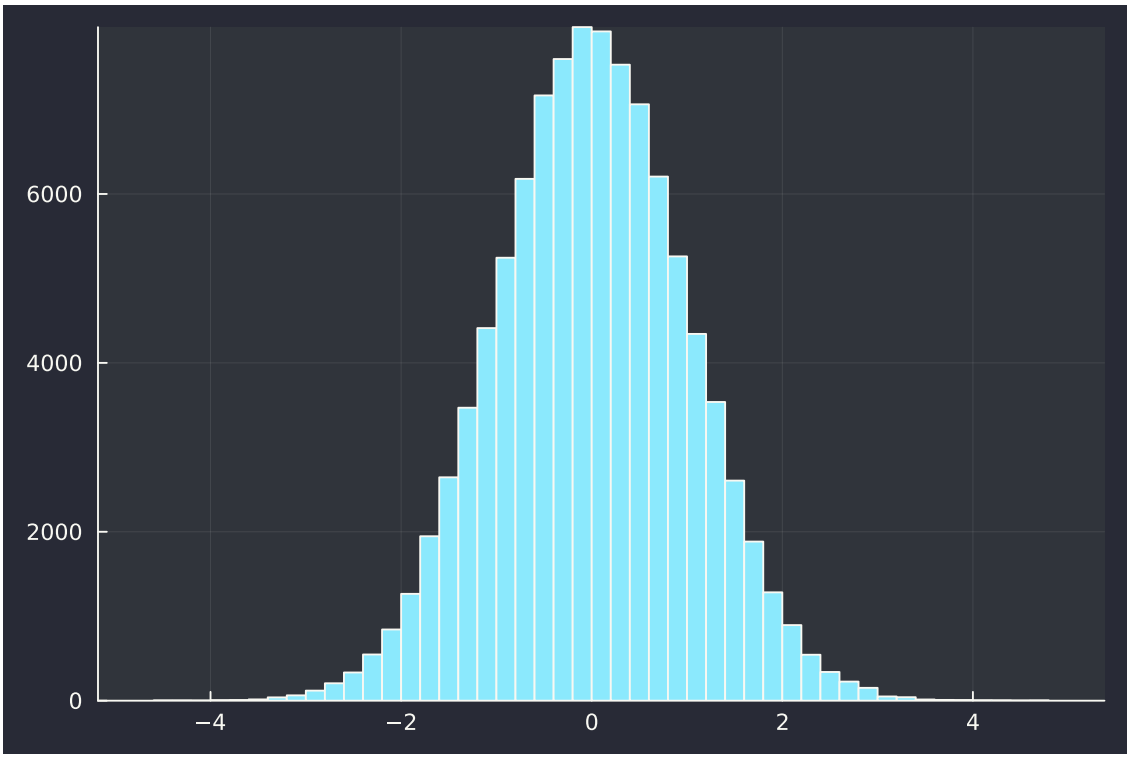
[88]:

Normal Distribution

[89]:
```
?randn;
```

[90]:
```
x = randn(10^5)
histogram(x, bins=64, legend=false)
```

[90]:



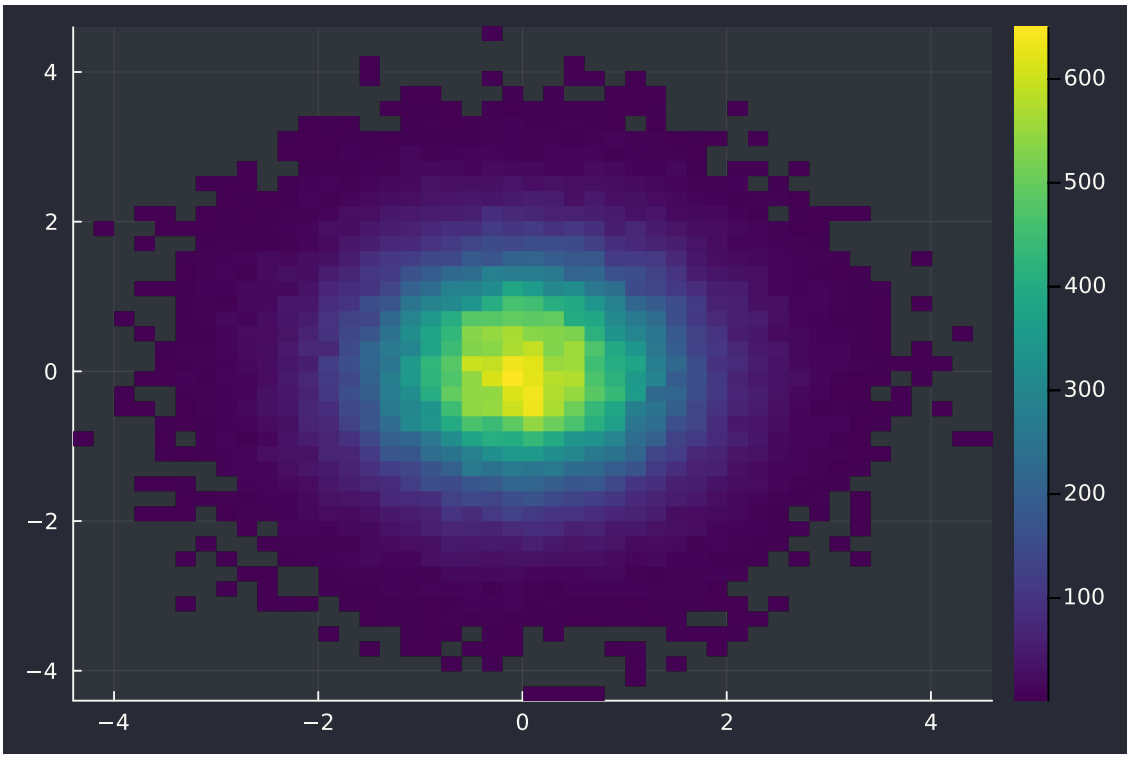
[91]:
```
md"**Histogram in 2D**"
```

[91]:

Histogram in 2D

[92]:
```
x = randn(10^5)
y = randn(10^5)
histogram2d(x, y, bins=(64, 64))
```

[92]:



3.0.6 Importing Data

Bring data from external files into `Julia`.

Data is typically stored in files, such as *CSV* or *JSON* files. In order to train and test machine learning models, the data needs to be loaded into the program. Additionally, the results of the training and testing process, such as model weights and performance metrics, also need to be saved to files. Therefore, the ability to manipulate files is essential for loading and saving data and model information in the machine learning process.

[93]:
```
using Pkg
Pkg.add("DataFrames")
Pkg.add("CSV")
```

```
   Resolving package versions...
  No Changes to `~/MEGA/git-repos/infodev/Codes/Project.toml`
  No Changes to `~/MEGA/git-
repos/infodev/Codes/Manifest.toml`
   Resolving package versions...
  No Changes to `~/MEGA/git-repos/infodev/Codes/Project.toml`
  No Changes to `~/MEGA/git-
repos/infodev/Codes/Manifest.toml`
```

[94]:
```
md"Create new CSV file"
```

[94]:
Create new CSV file

[95]:
```
using CSV, DataFrames
```

[96]:
```
md"`touch` command allows to create a file if it doesn't exist. Otherwise, it changes the file timestamps."
```

[96]:
`touch` command allows to create a file if it doesn't exist. Otherwise, it changes the file timestamps.

[97]:
```
touch("test-file.csv")
```

[97]:
```
"test-file.csv"
```

[98]:
```
;ls -la test-file.csv
```

```
-rw-rw-r-- 1 mhamdi mhamdi 66 Nov 29 21:36 test-file.csv
```

[99]:
```
file = open("test-file.csv", "w")
```

[99]:
```
IOStream(<file test-file.csv>)
```

[100]:
```
md"Let's create some imaginary data"
```

[100]:
Let's create some imaginary data

[101]:
```
df = DataFrame(
        Student = ["Mohamed", "Aymen", "Rami", "Ala"],
        Id = [1, 2, 3, 4],
        Marks = [18, 7, 12, 5.5]
        )
```

[101]:

	Student	Id	Marks
	String	Int64	Float64
1	Mohamed	1	18.0
2	Aymen	2	7.0
3	Rami	3	12.0
4	Ala	4	5.5

[102]:
```
md"Write `df` to file"
```

[102]:
Write `df` to file

[103]:
```
CSV.write("test-file.csv", df)
```

[103]:
```
"test-file.csv"
```

[104]:
```
md"Open the CSV file and add some contents. See what happens when we load it again."
```

[104]: Open the CSV file and add some contents. See what happens when we load it again.

[105]:
```
CSV.read("test-file.csv", DataFrame)
```

[105]:

	Student	Id	Marks
	String7	Int64	Float64
1	Mohamed	1	18.0
2	Aymen	2	7.0
3	Rami	3	12.0
4	Ala	4	5.5

3.0.7 Logical Arrays

Use logical expressions to help extracting elements of interest from `Julia` arrays.

[106]:
```
x = [1, 2, -5, 7.2, 3im]
println(x)
typeof(x)
```

```
ComplexF64[1.0 + 0.0im, 2.0 + 0.0im, -5.0 + 0.0im, 7.2 + 0.0im, 0.
↪0 + 3.0im]
```

[106]:
```
Vector{ComplexF64} (alias for Array{Complex{Float64},
1})
```

[107]:
```
idx = [false, true, false, false, true]
print(x[idx])
```

```
ComplexF64[2.0 + 0.0im, 0.0 + 3.0im]
```

[108]:
```
M = Array{Float64, 2}(undef, 5, 4)
```

[108]:
```
5×4 Matrix{Float64}:
 6.90677e-310  6.90677e-310  6.90677e-310  6.90676e-310
 6.90677e-310  6.90677e-310  6.90677e-310  6.90676e-310
 6.90677e-310  6.90677e-310  6.90676e-310  6.90676e-310
 6.90677e-310  6.90677e-310  6.90676e-310  6.90676e-310
 6.90677e-310  6.90677e-310  6.90677e-310  6.90677e-310
```

[109]:
```
row_idx = [true, false, true, true, false];
col_idx = [false, true, true, false];
```

[110]:
```
M[row_idx, :]
```

[110]:
```
3×4 Matrix{Float64}:
 6.90677e-310  6.90677e-310  6.90677e-310  6.90676e-310
 6.90677e-310  6.90677e-310  6.90676e-310  6.90676e-310
 6.90677e-310  6.90677e-310  6.90676e-310  6.90676e-310
```

[111]:
```
M[:, col_idx]
```

[111]:
```
5×2 Matrix{Float64}:
 6.90677e-310  6.90677e-310
 6.90677e-310  6.90677e-310
 6.90677e-310  6.90676e-310
 6.90677e-310  6.90676e-310
 6.90677e-310  6.90677e-310
```

[112]:
```
M[row_idx, col_idx]
```

[112]:
```
3×2 Matrix{Float64}:
 6.90677e-310  6.90677e-310
 6.90677e-310  6.90676e-310
 6.90677e-310  6.90676e-310
```

3.0.8 Programming

Write programs that execute code based on some condition.

[113]:
```
md"**Conditional Evaluation**"
```

[113]:

Conditional Evaluation

[114]:
```
a, b = □, □
if a < b
    println("$a is less than $b")
elseif a > b
    println("$a is greater than $b")
else
    println("$a is equal to $b")
end
```

□ is equal to □

[115]:
```
md"**`While` Loop**"
```

[115]:

`While` Loop

[116]:
```
fruits = ["Blueberry", "Orange", "Banana", "Raspberry",□
 ↪"Strawberry"]
iter = 1
while iter □ length(fruits)
    println("Item #$iter is $(fruits[iter])")
    iter +=1
end
```

```
Item #1 is Blueberry
Item #2 is Orange
Item #3 is Banana
Item #4 is Raspberry
Item #5 is Strawberry
```

[117]:
```
md"***`For` Loop**"
```

[117]:

For Loop

[118]:
```
vegetables = ["Broccoli", "Garlic", "Mushrooms", "Potatoes",
 ↪"Tomatoes"]
i = 1
for item in vegetables
    println("Item #$i is $item")
    i += 1
end
```

```
Item #1 is Broccoli
Item #2 is Garlic
Item #3 is Mushrooms
Item #4 is Potatoes
Item #5 is Tomatoes
```

3.0.9 Final Project

Bring together concepts that you have learned with a project.

This simple project consists of implementing a basic calculator. This latter could have the ability to perform basic arithmetic operations like *addition*, *subtraction*, *multiplication*, and *division*.

Here are the steps to be followed: 1. Create a function called `calculator()` that takes two arguments, `x` and `y`, and a char operation that specifies which operation to perform. 1. Use an `if-else` statement to check the value of operation. Depending on the value of operation, call the appropriate function to perform the calculation. 1. Test the calculator function by calling it with different values for `x`, `y`, and operation and printing the result. 1. Once the basic calculator is working, we can improve it by adding more functionality such as handling decimals and negative numbers, or implementing more advanced operations such as square root, power, trigonometry and so on. 1. Finally, we could also experiment with different input types, such as command line arguments or a graphical user interface.

[119]:
```
md"Here is an example of how the basic calculator function could
 ↪look like:"
```

[119]:

Here is an example of how the basic calculator function could look like:

[120]:
```
function calculator(x::Number, y::Number, op::Char)
    if op == '+'
        return x + y
    elseif op == '-'
```

```
        return x - y
    elseif op == '*'
        return x * y
    elseif op in ['/', '÷']
        return x / y
    else
        return "INVALID OPERATION"
    end
end
```

[120]: calculator (generic function with 1 method)

[121]:
```
println("Summation is $(calculator(5, 3, '+'))")
println("Subtraction is $(calculator(5, 3, '-'))")
println("Multiplication is $(calculator(5, 3, '*'))")
println("Division is $(calculator(5, 3, '÷'))")
println(calculator(5, 3, '×'))
```

```
Summation is 8
Subtraction is 2
Multiplication is 15
Division is 1.6666666666666667
INVALID OPERATION
```

Miscallenous

[122]:
```
md"Check your version of Julia"
versioninfo()
```

```
Julia Version 1.9.3
Commit bed2cd540a (2023-08-24 14:43 UTC)
Build Info:

  Note: This is an unofficial build, please report bugs to the project
  responsible for this build and not to the Julia project unless you can
  reproduce the issue using official builds available at
https://julialang.org/downloads

Platform Info:
  OS: Linux (x86_64-linux-gnu)
  CPU: 8 × Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-14.0.6 (ORCJIT, skylake)
  Threads: 2 on 8 virtual cores
```

[123]:
```
md"The macro `@edit` shows the defintion of a function when
↪invoked with specific arguments"
# @edit maximum([-1, 0, 1])
```

[123]:

The macro `@edit` shows the defintion of a function when invoked with specific arguments

[124]:
```
md"`varinfo` lists all global variables and their corresponding
↪types in the current scope"
varinfo()
```

[124]:

name	size	summary
Base		Module
Core		Module
Foo	0 bytes	Foo (generic function with 1 method)
M	200 bytes	5×4 Matrix{Float64}
Main		Module
PLOTS_DEFAULTS	456 bytes	Dict{Symbol, Symbol} with 1 entry
a	0 bytes	Irrational{:π}
b	0 bytes	Irrational{:π}
bar	0 bytes	bar (generic function with 1 method)
calculator	0 bytes	calculator (generic function with 1 method)
col_idx	44 bytes	4-element Vector{Bool}
df	915 bytes	4×3 DataFrame
dict	503 bytes	Dict{String, Int64} with 3 entries
f	0 bytes	f (generic function with 4 methods)
file	372 bytes	IOStream
foo	0 bytes	foo (generic function with 4 methods)
fruits	160 bytes	5-element Vector{String}
g	0 bytes	g (generic function with 1 method)
h	0 bytes	h (generic function with 1 method)
i	8 bytes	Int64
idx	45 bytes	5-element Vector{Bool}
iter	8 bytes	Int64
lst	176 bytes	5-element Vector{Any}
mycos	0 bytes	mycos (generic function with 2 methods)
row_idx	45 bytes	5-element Vector{Bool}
showall	0 bytes	showall (generic function with 1 method)
str	19 bytes	11-codeunit String
tmp	1 byte	UInt8
var	80 bytes	2×4 BitMatrix
vegetables	159 bytes	5-element Vector{String}
x	120 bytes	5-element Vector{ComplexF64}
y	781.289 KiB	100000-element Vector{Float64}
z	768 bytes	91-element Vector{Float64}

Modules

[125]:
```
module MyModule
export a
a = 0
b = true
end
```

[125]: Main.MyModule

[126]:
```
varinfo(MyModule)
```

[126]:

name	size	summary
MyModule	2.304 KiB	Module
a	8 bytes	Int64

[127]:
```
a
```

[127]: □ = 3.1415926535897…

[128]:
```
MyModule.a
```

[128]: 0

[129]:
```
MyModule.b
```

[129]: true

[ ]:
```

```