Introduction to Julia for AI

Prof. Paulo André Castro
pauloac@ita.br
www.comp.ita.br/~pauloac
Sala 110, Computer Science Division
ITA

Main Languages used in AI (alphabetic order)

- C /C++
- Haskell
- Java
- Julia
- LISP
- Prolog
- Python
- R-language
- but there are others...and There are also *Zillions* of frameworks/libraries
  - Some of them available in more than one language

Why Julia?

- *"We want a language that's open-source, with a liberal license. We want the **speed** of **C** with the **dynamism** of **Ruby**. We want a language that's homoiconic, with true macros like **Lisp**, but with an obvious, **familiar mathematical notation** like **Matlab**. We want something as usable for general programming as **Python**, as easy for **statistics** as **R,** as natural for string processing as **Perl**, as powerful for **linear algebra** as **Matlab**, as good at **glueing programs** together as the **shell**. Something that is dirt simple to learn yet keeps the most serious hackers happy. We want it interactive and we want it compiled."Julia's authors*

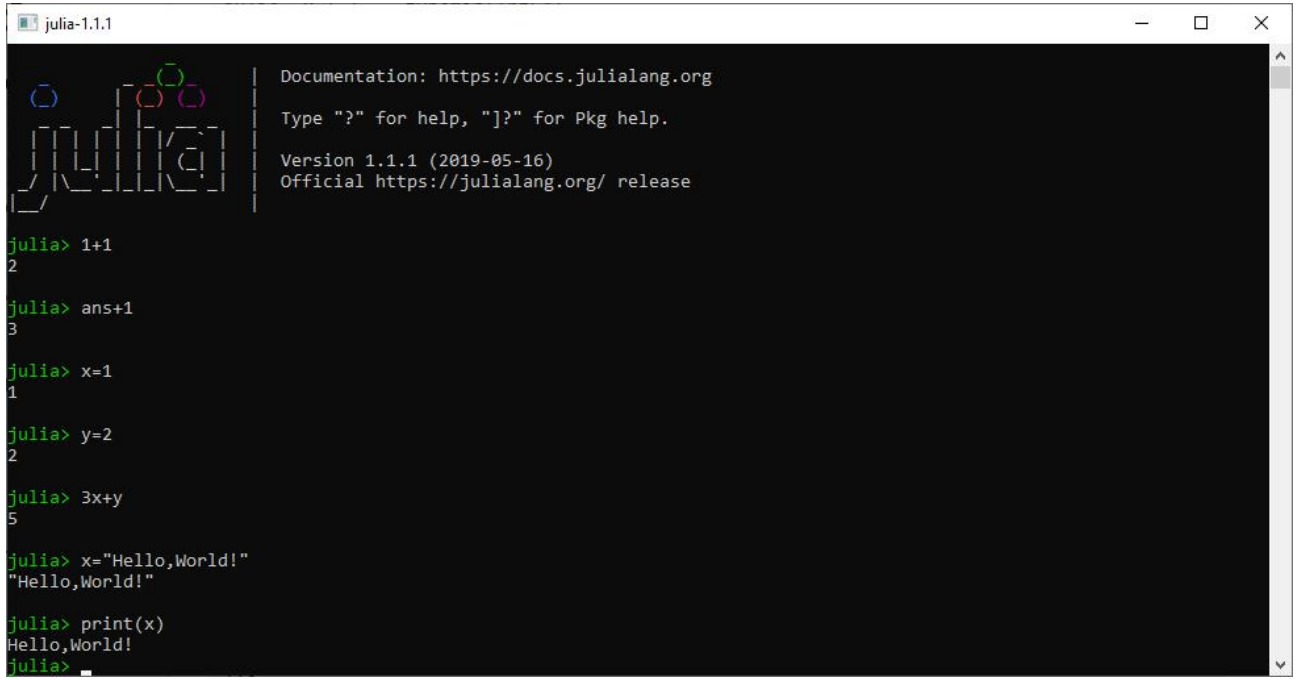
- Viral B Shah, Jeff Bezanson, Stefan Karpinski, Deepak Vinchhi, Keno Fischer and Alan Edelman founded Julia,
  - Available in 2012
  - Julia made its official debut in 2018 when Julia 1.0 was released

- *Main Site : https://julialang.org/*

Introduction to Julia Language

- **Basics**
- Arrays
- Control Flow
- Functions
- Plotting
- Data Files (csv) and Data Frames
- Random numbers and statistics

Interactive Session (REPL=Read-Eval-Print Loop)



Creating variables

```
# The symbol # starts a comment!
# Assign the value 10 to the variable x
julia> x = 10
# Doing math with x's value
julia> x + 111
# Reassign x's value
julia> x = 1 + 12
# You can assign values of other types, like strings of text
julia> x = "Hello World!"
julia> x = 1.0
julia> y = -3

# Julia is dynamically typed, but in well-written julia code the types can usually be
#inferred. You often get a major performance enhancement when that is possible.
```

Stylistic Conventions

- Names of variables are in lower case.
- Word separation can be indicated by underscores ('_'), but use of underscores is discouraged unless the name would be hard to read otherwise
- Names of Types and Modules begin with a capital letter and word separation is shown with upper camel case instead of underscores
- Names of functions and macros are in lower case, without underscores
- Functions that write to their arguments have names that end in !
  - These are sometimes called "mutating" or "in-place" functions because they are intended to produce changes in their arguments after the function is called, not just return a value.

Variables with Types

```
julia>x=1
1
julia>typeof(x)
Int64
julia>x=1.0
1.0
julia>typeof(x)
Float64
julia>typeof(convert(Float32,x))
Float32
• julia>x=Float64(0)
0.0
julia>typeof(x)
Float64
```

```
julia>1/0
Inf
julia>convert(Int64,1.5)
Error!!!
# Use round, floor or ceil
julia> round(1.5)
2
julia> round(1.4)
1.0
#Machine Epsilon
julia> eps(Float32)
```

Integer Types

Type	Signed?	Number of bits	Smallest value	Largest value
Int8	✓	8	-2^7	2^7 - 1
UInt8		8	0	2^8 - 1
Int16	✓	16	-2^15	2^15 - 1
UInt16		16	0	2^16 - 1
Int32	✓	32	-2^31	2^31 - 1
UInt32		32	0	2^32 - 1
Int64	✓	64	-2^63	2^63 - 1
UInt64		64	0	2^64 - 1
Int128	✓	128	-2^127	2^127 - 1
UInt128		128	0	2^128 - 1
Bool	N/A	8	false (0)	true (1)

Floating-point Types

Type	Precision	Number of bits
Float16	half	16
Float32	single	32
Float64	double	64

Getting Help...?

```
julia>?convert<ENTER>
search: convert code_native @code_native
  convert(T, x)
  Convert x to a value of type T.
  If T is an Integer type, an InexactError will be raised if x is not representable by T, for
example if x is not integer-valued, or is outside the range supported by T.
  Examples
  ==========
  julia> convert(Int, 3.0)
  3
julia> convert(Int, 3.5)
  ERROR: InexactError: Int64(3.5)
  Stacktrace:
  [...]  ............................
```

Basic Control Flow

 # if
if *condition*
    cmds
[**elseif** *condition*
    cmds]
[**else**
    cmds]
end

- # Example

```
julia>if x > 5
  print("x is bigger than five")
end
```

- # For

```
julia> for i in 1:5
            println("line",i)
        end
```

Basic Operations

x+y, x-y, x*y, x/y	As expected	
x /y	divide	(truncated to an integer, if both int)
x \ y	inverse divide	equivalent to y / x
x ^ y	power	raises x to the yth power
x % y	remainder	equivalent to rem(x,y)

Loading and running a Julia file

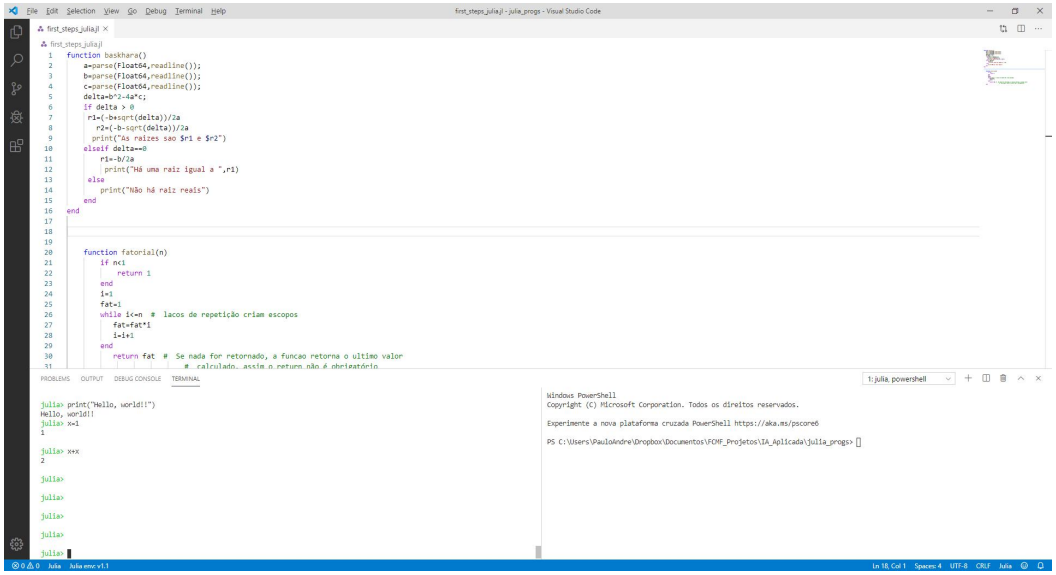
```
# The symbol # starts a comment!
# load and run a .jl file (julia program) from #prompt

julia>include("file.jl");

# # load and run a .jl file (julia program)
# from OS command line
julia script.jl arg1 arg2
```

Using Julia with an Editor

- There are many Editor and IDEs available for Julia
- We suggest a simple one (Visual Studio Code or notepad++ or any other that you like)
- In VS code, press Shit+Enter to open Terminal (bottom of the image above)



Introduction to Julia Language

- Basics
- **Arrays**
- Control Flow
- Functions
- Plotting
- Data Files (csv) and Data Frames

Arrays

```
julia> [ 1 , 2, 3]
3-element Array{Int64,1}
1
2
3
julia> [ 1, 2,3].^3  # dot call (each element)
3-element Array{Int64,1}
1
8
27
```

Arrays - 2

```
julia> zeros(Int64,3,3)# zeros(type, dimensions)
3×3 Array{Int64,2}:
 0  0  0
 0  0  0
 0  0  0

julia> ones(Int64,2,2)  # guess ?
julia> A=ones(Int64,2,2); B=ones(Int64,2,2)
A*B
A+B
A^2+B^3+5*A
```

Arrays -3

```
julia> vcat([1, 2], 3)
3-element Array{Int64,1}:
 1
 2
 3
julia> hcat([1 2], 3)
1×3 Array{Int64,2}:
 1  2  3
```

Math with arrays

```
julia>a=[1 2 3]; b=[3 2 1]
julia>c=a+b
[4 4 4]
julia>a*c  # Error dimension mismatch
julia>a*b' # a* transpose of b
10
julia>a3p3=ones(Int64,3,3) # creates matrix 3x3 with   #ones
julia>b3p3=a3p3.+4# creates 3x3 adding 4 for each element
5  5  5
5  5  5
5  5  5
julia> a3p3*b3p3 # product of matrixes
15  15  15
15  15  15
15  15  15
```

Basic Linear Algebra

```
julia>A= [ 1 2 3 ; 4 1 6; 7 7 7]
1 2 3
4 1 6
7 7 7
julia>tr(A)  # matrix trace (sum of diagonal elements)
9
julia>det(A)  # matrix determinant
56.0
julia>inv(A)  # matrix inverse
-0.625   0.125   0.160714
 0.25   -0.25    0.107143
 0.375   0.125  -0.125
julia> eigvals(A) # eigen values
julia> eigvecs(A) # eigen vectors
julia>lu(A)  # return L -U matrixes from LU factorization
```

Dictionaries

- A Dictionary is a very useful Data structure that stores a set of unique ids (keys) to identify objects (values). In julia, you may define what types will be used for keys and objects. Examples:

```
julia> D = Dict('a'=>2, 'b'=>3)
Dict{Char,Int64} with 2 entries:
  'a' => 2
  'b' => 3
julia>d['b']
3
# returns true if dict has the given key, false otherwise
julia> haskey(D, 'a')
 true
julia> d = Dict("a"=>1, "b"=>2); # ; supress the output

julia> get(d, "a", 3)  # returns d["a"] or 3 if "a" is a not a valid key
1
```

Dictionaries – 2

- #also creates a dict.

```
julia>d=Dict([("A", 1), ("B", 2)])
Dict{String,Int64} with 2 entries:
  "B" => 2
  "A" => 1
julia>push!(d,”c”=>3)
Dict{String,Int64} with 2 entries:
  "C" =>3
  "B" => 2
  "A" => 1
julia> pop!(d) # remove the last inserted element
julia>pop!(d,”A”) # remove the element with “A” key
```

Introduction to Julia Language

- Basics
- Arrays
- **Control Flow**
- Functions
- Plotting
- Data Files (csv) and Data Frames

Control Flow

```
if x < y
    println("x is less than y")
elseif x > y  # elseif is optional
    println("x is greater than y")
else   # else is also optional
    println("x is equal to y")
end    # end is not optional
# ternary operators are ok
> x = 1; y = 2;
> println(x < y ? "less than" : "not less than")
less than
```

Control Flow – While

```
julia> i = 1; # global variable
julia> while i <= 5
          println(i)
          global i += 1# refers to same i, it
       #does not create a new
      end
1
2
3
4
5
```

Control Flow – For

```
julia> for i = 1:5
          println(i)
       end
1
2
3
4
5
```

```
# for and arrays
>for i in [1,4,0]
    println(i)
end
1
4
0
```

Control flow – For

```
julia> using Random
julia>Random.seed!(0)
julia>x=rand(100) # array with 100 random elements
julia> s=0
julia> for i = eachindex(x)# it sums all elements
          global s+= x[i] # for creates a new
      #scope, to refer to global scope use 'global'
       end
```

Introduction to Julia Language

- Basics
- Arrays
- Control Flow
- **Functions**
- Plotting
- Data Files (csv) and Data Frames

Functions -1

```
julia> function f(x,y)  # creates a function f
          x + y
       end
julia> f(x,y)=x+y  # also creates a function f

julia> f(2,3) # calls function f
5
julia> g= f;  #without parentheses f is the function
julia>g(1,1)
2
```

Functions – 2

- Julia function arguments are not copied when they are passed to functions
- Function arguments themselves act as new variable bindings (new locations that can refer to values), but the values they refer to are identical to the passed values
- Modifications to **mutable values** (such as Arrays) made within a function will be visible to the caller
- You can use Return, if not the function returns the last expression value. Return does not finish the function!!

Functions – 3

```
julia> f(x,y) = x + y
f (generic function with 1 method)

julia> function g(x,y)
         return x * y
         x + y
       end
g (generic function with 1 method)

julia> f(2,3)
5

julia> g(2,3)
6
```

Functions – 4

- You can use map to call a function for each element of an array

```
julia> map(round, [1.2,3.5,1.7])
3-element Array{Float64,1}:
 1.0
 4.0
 2.0
```

- anonymous functions can also be created

```
julia>map(x -> x^2 + 2x - 1, [1,3,-1])
3-element Array{Int64,1}:
  2
 14
 -2
```

Functions – 5

- Mutating functions. It is common convention to use ! in functions that mutates its arguments

```
julia>=[1 2]
julia> push!(a,3,4) # it puts 3 and 4 in a
```

Introduction to Julia Language

- Basics
- Arrays
- Control Flow
- Functions
- **Plotting**
- Data Files (csv) and Data Frames

Plotting

```
using Pkg
Pkg.add("PyPlot")
using PyPlot
x = range(0,stop=2*pi,length=1000);
y = sin.(3*x + 4*cos.(2*x))
plot(x, y, color="red", linewidth=2.0, linestyle="--")
```

Result

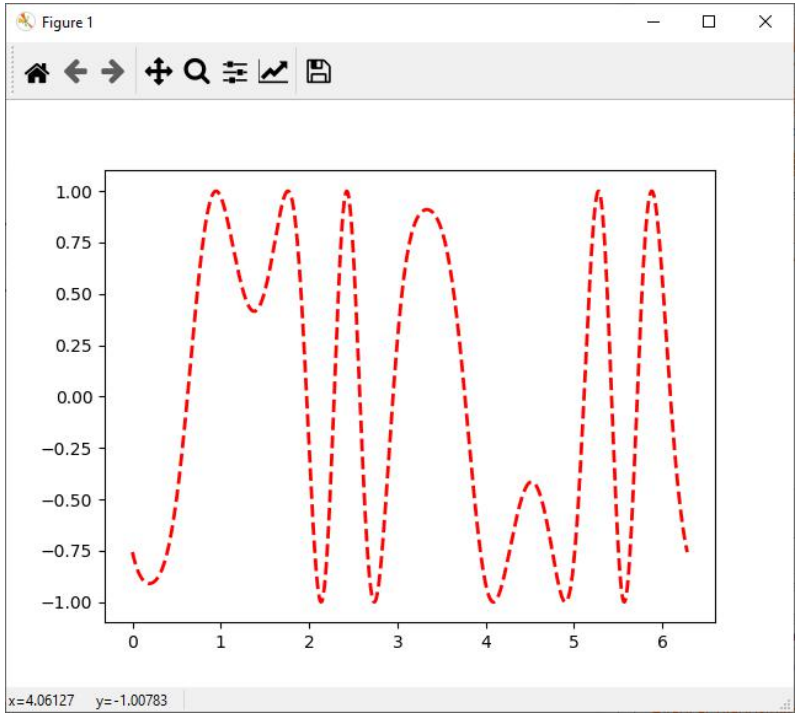


Figure 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6
x=4.06127 y=-1.00783

Introduction to Julia Language

- Basics
- Arrays
- Control Flow
- Functions
- Plotting
- **Data Files (csv) and Data Frames**

Loading CSV files with DataFrames

```
using CSV;
using DataFrames

df=CSV.read("file",delim=',')
#getting the type of  df
typeof(df)
#accessing position (1,1). DataFrames also # starts at 1 in
#julia
df[1,1]
```

Handling DataFrames

```
# returns columns and first rows
head(df)
# returns columns and last rows
tail(df)
# returns (number of rows, number of colums)
size(df)

size(df)[1] # number of rows

size(df)[2] # number of columns
```

Introduction to Julia Language

- Basics
- Arrays
- Control Flow
- Functions
- Plotting
- Data Files (csv) and Data Frames
- **Random numbers and statistics**

Random number and statistics

- randn(100) # return 100 numbers sampled from a normal distribution with mean 0 and standard deviation of 1

Plotting a Histogram

```
using PyPlot
#  Create Data  #
x = randn(1000) # Values
nbins = 50 # Number of bins (columns)

#  Plot
fig = figure("pyplot_histogram",figsize=(10,10)) # Not strictly required
ax = PyPlot.axes() # Not strictly required
h = plt.hist(x,nbins) # Histogram

grid("on")
xlabel("X")
ylabel("Y")
title("Histograma")
```