julia_Final

June 18, 2018

- se lanzó su primera versión en el 2012.
- sus creadores son Stefan Karpinski, Viral Shah, Alan Edelman, Jeff Bezanson.
- Código abierto, licenciado por el MIT.
- Para Enero del 2018 tenía 1.800.000

1 Para que sirve

- Apunta a la comunidad cientifca.
- Gran manejo matemático.
- Suele utilizarse para programación paralela y distributiva.
- Es muy utilizado en el ámbito de machine learning.

2 Caracteristicas basicas del lenguaje

- Tipado dinámico con beneficios de tipado estatico.
- Las librerías de python, c y fortran están a un llamado de distancia.
- Dispone de un compilador avanzado, JIT (Just In Time).
- Busca la velocidad tanto en tiempo de desarrollo como de ejecución.

2.0.1 Variables

```
In [1]: miVariable = 1
```

```
Out[1]: 1
```

```
In [2]: miVariable = miVariable + 5
```

```
Out[2]: 6
```

```
In [3]:  = 0.00001
```

```
Out[3]: 1.0e-5

In [4]:  = "Hola a todos y todas"

Out[4]: "Hola a todos y todas"

In [5]: # Constantes matemáticas


Out[5]:  = 3.1415926535897...

In [6]: miVariable = 2* + 0.12

Out[6]: 6.403185307179586

In [7]: miVariable = miVariable + Inf

Out[7]: Inf

In [8]: miVariable = miVariable + -Inf

Out[8]: NaN

In [9]: # soporte para números irracionales
        res = (1 + 4im)*(2 - 3im)

Out[9]: 14 + 5im

In [10]: cos(res)

Out[10]: 10.147261924626015 - 73.50624621453295im

In [11]:  = 3.14

WARNING: imported binding for  overwritten in module Main


Out[11]: 3.14

In [13]: tup = (1, 2, 3)
         length(tup)

Out[13]: 3

In [14]: in(4, tup) # => true

Out[14]: false

In [15]: d, e, f = 4, 5, 6

Out[15]: (4,5,6)
```

```
In [16]: e, d = d, e
         e, d
```

```
Out[16]: (4,5)
```

```
In [17]: # números racionales
         1//3
```

```
Out[17]: 1//3
```

```
In [18]: float(1//3)
```

```
Out[18]: 0.3333333333333333
```

2.0.2 Tipado

- En Julia los tipados son, por defecto, omitidos.
- Los valores tienen tipo.
- Las variables son simples notaciones que hacen referencia a entidades.
- Es fácil expresar el tipo esperado del valor de una cierta variable.

```
In [15]: x = "Soy un string"
```

```
Out[15]: "Soy un string"
```

```
In [16]: y::Int64
         y = x
```

```
        TypeError: typeassert: expected Int64, got ASCIIString
```

2.0.3 Manejo de errores simple

```
In [1]: tup[1] # => 1
        try:
            tup[1] = 3 # MethodError
        catch e
            println(e)
        end
```

```
        UndefVarError: tup not defined
```

```
In [1]: ampliacion_sqrt(x) =

        x >= 0 ? sqrt(x) : error("Valor de x negativo es inválido")

        ampliacion_sqrt(100)
```

```
Out[1]: 10.0
```

```
In [2]: ampliacion_sqrt(-6)
```

```
        Valor de x negativo es inválido



         in ampliacion_sqrt at ./In[1]:1
```

2.0.4 Funciones y argumentos

Los argumentos en Julia se pasan siguiendo una convención que suele llamarse "pass-by-sharing".

```
In [3]: (x -> x^2 + 2x - 1)(2)
```

```
Out[3]: 7
```

```
In [19]: map(x -> x^2 + 2x - 1, [1,3,-1])
```

```
Out[19]: 3-element Array{Int64,1}:
           2
          14
          -2
```

```
In [20]: function foo(a,b)
                     a+b, a*b
         end
         x, y = foo(2,3)
```

```
Out[20]: (5,6)
```

```
In [21]: function default(a,b,x=5,y=6)
             return "$a $b and $x $y"
         end

         default('h','g')
```

```
Out[21]: "h g and 5 6"
```

```
In [22]: default('h', 'g', 'i')
```

```
Out[22]: "h g and i 6"
```

2.0.5 Duck Typing

Quiero mostrar que a julia no le importa el tipo de dato. Julia opera con cualquier tipo de dato que tenga sentido

```
In [1]: f2(x) = x^2

Out[1]: f2 (generic function with 1 method)

In [3]: f2(2)

Out[3]: 4

In [4]: Ad = rand(3, 3)
        f2(Ad)

Out[4]: 3Œ3 Array{Float64,2}:
         1.23116  0.915372  0.746678
         1.50719  1.21557   0.948403
         1.52776  1.18283   1.00988

In [6]: f2("hola") #La multiplicacion de este parametro implica una concatenacion

Out[6]: "holahola"

In [7]: v = rand(3)
        f2(v) # DimensionMismatch


        DimensionMismatch("Cannot multiply two vectors")



        Stacktrace:

         [1] power_by_squaring(::Array{Float64,1}, ::Int64) at ./intfuncs.jl:169

         [2] f2(::Array{Float64,1}) at ./In[1]:1
```

2.0.6 Funciones mutantes vs no mutantes

Por convencion las funciones seguidas por un ! alteran, o bien mutan, sus contenidos y las que carecen de un ! no lo hacen

```
In [8]: v = [4, 7, 1]

Out[8]: 3-element Array{Int64,1}:
         4
         7
         1
```

```
In [9]: sort(v)

Out[9]: 3-element Array{Int64,1}:
         1
         4
         7

In [10]: v

Out[10]: 3-element Array{Int64,1}:
          4
          7
          1

In [11]: sort!(v)

Out[11]: 3-element Array{Int64,1}:
          1
          4
          7

In [12]: v

Out[12]: 3-element Array{Int64,1}:
          1
          4
          7
```

2.0.7 Broadcasting

Si ponemos un . entre el nombre de la funcion y su lista de argumento le estamos diciendo a la funcion que se aplique sobre cada elemento sobre cada elemento del input. Esto es nativo de Julia y sirve para cualquier funcion.

```
In [13]: Am = [i + 3*j for j in 0:2, i in 1:3]

Out[13]: 3Œ3 Array{Int64,2}:
          1  2  3
          4  5  6
          7  8  9

In [14]: f2(Am)  #Esto seria A~2 = A*A

Out[14]: 3Œ3 Array{Int64,2}:
           30   36   42
           66   81   96
          102  126  150

In [16]: f2.(Am) #Esto en cambio aplica x~2 a cada elemento de la matriz

Out[16]: 3Œ3 Array{Int64,2}:
           1   4   9
          16  25  36
          49  64  81
```

2.0.8 Multiple Dispatch

rapido, extensible, proglamable facilmente

In [18]:
```
#Para entender el despacho multiple en Julia, observemos el operador +
#Si llamamos a la funcion methods() sobre +, podemos ver todas las definiciones de +
methods(+)
```

Out[18]:
```
# 180 methods for generic function "+":
+(x::Bool, z::Complex{Bool}) in Base at complex.jl:232
+(x::Bool, y::Bool) in Base at bool.jl:89
+(x::Bool) in Base at bool.jl:86
+(x::Bool, y::T) where T<:AbstractFloat in Base at bool.jl:96
+(x::Bool, z::Complex) in Base at complex.jl:239
+(a::Float16, b::Float16) in Base at float.jl:372
+(x::Float32, y::Float32) in Base at float.jl:374
+(x::Float64, y::Float64) in Base at float.jl:375
+(z::Complex{Bool}, x::Bool) in Base at complex.jl:233
+(z::Complex{Bool}, x::Real) in Base at complex.jl:247
+(x::Char, y::Integer) in Base at char.jl:40
+(c::BigInt, x::BigFloat) in Base.MPFR at mpfr.jl:312
+(a::BigInt, b::BigInt, c::BigInt, d::BigInt, e::BigInt) in Base.GMP at gmp.jl:334
+(a::BigInt, b::BigInt, c::BigInt, d::BigInt) in Base.GMP at gmp.jl:327
+(a::BigInt, b::BigInt, c::BigInt) in Base.GMP at gmp.jl:321
+(x::BigInt, y::BigInt) in Base.GMP at gmp.jl:289
+(x::BigInt, c::Union{UInt16, UInt32, UInt64, UInt8}) in Base.GMP at gmp.jl:346
+(x::BigInt, c::Union{Int16, Int32, Int64, Int8}) in Base.GMP at gmp.jl:362
+(a::BigFloat, b::BigFloat, c::BigFloat, d::BigFloat, e::BigFloat) in Base.MPFR at mp
+(a::BigFloat, b::BigFloat, c::BigFloat, d::BigFloat) in Base.MPFR at mpfr.jl:453
+(a::BigFloat, b::BigFloat, c::BigFloat) in Base.MPFR at mpfr.jl:447
+(x::BigFloat, c::BigInt) in Base.MPFR at mpfr.jl:308
+(x::BigFloat, y::BigFloat) in Base.MPFR at mpfr.jl:277
+(x::BigFloat, c::Union{UInt16, UInt32, UInt64, UInt8}) in Base.MPFR at mpfr.jl:284
+(x::BigFloat, c::Union{Int16, Int32, Int64, Int8}) in Base.MPFR at mpfr.jl:292
+(x::BigFloat, c::Union{Float16, Float32, Float64}) in Base.MPFR at mpfr.jl:300
+(B::BitArray{2}, J::UniformScaling) in Base.LinAlg at linalg/uniformscaling.jl:59
+(a::Base.Pkg.Resolve.VersionWeights.VWPreBuildItem, b::Base.Pkg.Resolve.VersionWeigh
+(a::Base.Pkg.Resolve.VersionWeights.VWPreBuild, b::Base.Pkg.Resolve.VersionWeights.V
+(a::Base.Pkg.Resolve.VersionWeights.VersionWeight, b::Base.Pkg.Resolve.VersionWeight
+(a::Base.Pkg.Resolve.MaxSum.FieldValues.FieldValue, b::Base.Pkg.Resolve.MaxSum.Field
+(x::Base.Dates.CompoundPeriod, y::Base.Dates.CompoundPeriod) in Base.Dates at dates/
+(x::Base.Dates.CompoundPeriod, y::Base.Dates.Period) in Base.Dates at dates/periods.
+(x::Base.Dates.CompoundPeriod, y::Base.Dates.TimeType) in Base.Dates at dates/period
+(x::Date, y::Base.Dates.Day) in Base.Dates at dates/arithmetic.jl:77
+(x::Date, y::Base.Dates.Week) in Base.Dates at dates/arithmetic.jl:75
+(dt::Date, z::Base.Dates.Month) in Base.Dates at dates/arithmetic.jl:58
+(dt::Date, y::Base.Dates.Year) in Base.Dates at dates/arithmetic.jl:32
+(dt::Date, t::Base.Dates.Time) in Base.Dates at dates/arithmetic.jl:20
+(t::Base.Dates.Time, dt::Date) in Base.Dates at dates/arithmetic.jl:24
```

```
+(x::Base.Dates.Time, y::Base.Dates.TimePeriod) in Base.Dates at dates/arithmetic.jl:8
+(dt::DateTime, z::Base.Dates.Month) in Base.Dates at dates/arithmetic.jl:52
+(dt::DateTime, y::Base.Dates.Year) in Base.Dates at dates/arithmetic.jl:28
+(x::DateTime, y::Base.Dates.Period) in Base.Dates at dates/arithmetic.jl:79
+(y::AbstractFloat, x::Bool) in Base at bool.jl:98
+(x::T, y::T) where T<:Union{Int128, Int16, Int32, Int64, Int8, UInt128, UInt16, UInt3
+(x::Integer, y::Ptr) in Base at pointer.jl:128
+(z::Complex, w::Complex) in Base at complex.jl:221
+(z::Complex, x::Bool) in Base at complex.jl:240
+(x::Real, z::Complex{Bool}) in Base at complex.jl:246
+(x::Real, z::Complex) in Base at complex.jl:258
+(z::Complex, x::Real) in Base at complex.jl:259
+(x::Rational, y::Rational) in Base at rational.jl:245
+(x::Integer, y::Char) in Base at char.jl:41
+(i::Integer, index::CartesianIndex) in Base.IteratorsMD at multidimensional.jl:110
+(c::Union{UInt16, UInt32, UInt64, UInt8}, x::BigInt) in Base.GMP at gmp.jl:350
+(c::Union{Int16, Int32, Int64, Int8}, x::BigInt) in Base.GMP at gmp.jl:363
+(c::Union{UInt16, UInt32, UInt64, UInt8}, x::BigFloat) in Base.MPFR at mpfr.jl:288
+(c::Union{Int16, Int32, Int64, Int8}, x::BigFloat) in Base.MPFR at mpfr.jl:296
+(c::Union{Float16, Float32, Float64}, x::BigFloat) in Base.MPFR at mpfr.jl:304
+(x::Irrational, y::Irrational) in Base at irrationals.jl:109
+(x::Real, r::Base.Use_StepRangeLen_Instead) in Base at deprecated.jl:1232
+(x::Number) in Base at operators.jl:399
+(x::T, y::T) where T<:Number in Base at promotion.jl:335
+(x::Number, y::Number) in Base at promotion.jl:249
+(x::Real, r::AbstractUnitRange) in Base at range.jl:721
+(x::Number, r::AbstractUnitRange) in Base at range.jl:723
+(x::Number, r::StepRangeLen) in Base at range.jl:726
+(x::Number, r::LinSpace) in Base at range.jl:730
+(x::Number, r::Range) in Base at range.jl:724
+(r::Range, x::Number) in Base at range.jl:732
+(r1::OrdinalRange, r2::OrdinalRange) in Base at range.jl:882
+(r1::LinSpace{T}, r2::LinSpace{T}) where T in Base at range.jl:889
+(r1::StepRangeLen{T,R,S} where S, r2::StepRangeLen{T,R,S} where S) where {R<:Base.Twi
+(r1::StepRangeLen{T,S,S} where S, r2::StepRangeLen{T,S,S} where S) where {T, S} in Ba
+(r1::Union{LinSpace, OrdinalRange, StepRangeLen}, r2::Union{LinSpace, OrdinalRange, S
+(x::Base.TwicePrecision, y::Number) in Base at twiceprecision.jl:455
+(x::Number, y::Base.TwicePrecision) in Base at twiceprecision.jl:458
+(x::Base.TwicePrecision{T}, y::Base.TwicePrecision{T}) where T in Base at twiceprecis
+(x::Base.TwicePrecision, y::Base.TwicePrecision) in Base at twiceprecision.jl:465
+(x::Ptr, y::Integer) in Base at pointer.jl:126
+(A::BitArray, B::BitArray) in Base at bitarray.jl:1177
+(A::SymTridiagonal, B::SymTridiagonal) in Base.LinAlg at linalg/tridiag.jl:128
+(A::Tridiagonal, B::Tridiagonal) in Base.LinAlg at linalg/tridiag.jl:629
+(A::UpperTriangular, B::UpperTriangular) in Base.LinAlg at linalg/triangular.jl:419
+(A::LowerTriangular, B::LowerTriangular) in Base.LinAlg at linalg/triangular.jl:420
+(A::UpperTriangular, B::Base.LinAlg.UnitUpperTriangular) in Base.LinAlg at linalg/tri
+(A::LowerTriangular, B::Base.LinAlg.UnitLowerTriangular) in Base.LinAlg at linalg/tri
```

```
+(A::Base.LinAlg.UnitUpperTriangular, B::UpperTriangular) in Base.LinAlg at linalg/tri
+(A::Base.LinAlg.UnitLowerTriangular, B::LowerTriangular) in Base.LinAlg at linalg/tri
+(A::Base.LinAlg.UnitUpperTriangular, B::Base.LinAlg.UnitUpperTriangular) in Base.Lin
+(A::Base.LinAlg.UnitLowerTriangular, B::Base.LinAlg.UnitLowerTriangular) in Base.Lin
+(A::Base.LinAlg.AbstractTriangular, B::Base.LinAlg.AbstractTriangular) in Base.LinAl
+(A::Symmetric, x::Bool) in Base.LinAlg at linalg/symmetric.jl:274
+(A::Symmetric, x::Number) in Base.LinAlg at linalg/symmetric.jl:276
+(A::Hermitian, x::Bool) in Base.LinAlg at linalg/symmetric.jl:274
+(A::Hermitian, x::Real) in Base.LinAlg at linalg/symmetric.jl:276
+(Da::Diagonal, Db::Diagonal) in Base.LinAlg at linalg/diagonal.jl:140
+(A::Bidiagonal, B::Bidiagonal) in Base.LinAlg at linalg/bidiag.jl:330
+(UL::UpperTriangular, J::UniformScaling) in Base.LinAlg at linalg/uniformscaling.jl:
+(UL::Base.LinAlg.UnitUpperTriangular, J::UniformScaling) in Base.LinAlg at linalg/un
+(UL::LowerTriangular, J::UniformScaling) in Base.LinAlg at linalg/uniformscaling.jl:
+(UL::Base.LinAlg.UnitLowerTriangular, J::UniformScaling) in Base.LinAlg at linalg/un
+(A::Array, B::SparseMatrixCSC) in Base.SparseArrays at sparse/sparsematrix.jl:1541
+(x::Union{Base.ReshapedArray{T,1,A,MI} where MI<:Tuple{Vararg{Base.MultiplicativeInv
+(x::Union{Base.ReshapedArray{#s268,N,A,MI} where MI<:Tuple{Vararg{Base.Multiplicativ
+(A::SparseMatrixCSC, J::UniformScaling) in Base.SparseArrays at sparse/sparsematrix.
+(A::AbstractArray{TA,2}, J::UniformScaling{TJ}) where {TA, TJ} in Base.LinAlg at lin
+(A::Diagonal, B::Bidiagonal) in Base.LinAlg at linalg/special.jl:113
+(A::Bidiagonal, B::Diagonal) in Base.LinAlg at linalg/special.jl:114
+(A::Diagonal, B::Tridiagonal) in Base.LinAlg at linalg/special.jl:113
+(A::Tridiagonal, B::Diagonal) in Base.LinAlg at linalg/special.jl:114
+(A::Diagonal, B::Array{T,2} where T) in Base.LinAlg at linalg/special.jl:113
+(A::Array{T,2} where T, B::Diagonal) in Base.LinAlg at linalg/special.jl:114
+(A::Bidiagonal, B::Tridiagonal) in Base.LinAlg at linalg/special.jl:113
+(A::Tridiagonal, B::Bidiagonal) in Base.LinAlg at linalg/special.jl:114
+(A::Bidiagonal, B::Array{T,2} where T) in Base.LinAlg at linalg/special.jl:113
+(A::Array{T,2} where T, B::Bidiagonal) in Base.LinAlg at linalg/special.jl:114
+(A::Tridiagonal, B::Array{T,2} where T) in Base.LinAlg at linalg/special.jl:113
+(A::Array{T,2} where T, B::Tridiagonal) in Base.LinAlg at linalg/special.jl:114
+(A::SymTridiagonal, B::Tridiagonal) in Base.LinAlg at linalg/special.jl:122
+(A::Tridiagonal, B::SymTridiagonal) in Base.LinAlg at linalg/special.jl:123
+(A::SymTridiagonal, B::Array{T,2} where T) in Base.LinAlg at linalg/special.jl:122
+(A::Array{T,2} where T, B::SymTridiagonal) in Base.LinAlg at linalg/special.jl:123
+(A::Diagonal, B::SymTridiagonal) in Base.LinAlg at linalg/special.jl:131
+(A::SymTridiagonal, B::Diagonal) in Base.LinAlg at linalg/special.jl:132
+(A::Bidiagonal, B::SymTridiagonal) in Base.LinAlg at linalg/special.jl:131
+(A::SymTridiagonal, B::Bidiagonal) in Base.LinAlg at linalg/special.jl:132
+(A::Diagonal, B::UpperTriangular) in Base.LinAlg at linalg/special.jl:143
+(A::UpperTriangular, B::Diagonal) in Base.LinAlg at linalg/special.jl:144
+(A::Diagonal, B::Base.LinAlg.UnitUpperTriangular) in Base.LinAlg at linalg/special.j
+(A::Base.LinAlg.UnitUpperTriangular, B::Diagonal) in Base.LinAlg at linalg/special.j
+(A::Diagonal, B::LowerTriangular) in Base.LinAlg at linalg/special.jl:143
+(A::LowerTriangular, B::Diagonal) in Base.LinAlg at linalg/special.jl:144
+(A::Diagonal, B::Base.LinAlg.UnitLowerTriangular) in Base.LinAlg at linalg/special.j
+(A::Base.LinAlg.UnitLowerTriangular, B::Diagonal) in Base.LinAlg at linalg/special.j
```

```
+(A::Base.LinAlg.AbstractTriangular, B::SymTridiagonal) in Base.LinAlg at linalg/speci
+(A::SymTridiagonal, B::Base.LinAlg.AbstractTriangular) in Base.LinAlg at linalg/speci
+(A::Base.LinAlg.AbstractTriangular, B::Tridiagonal) in Base.LinAlg at linalg/special
+(A::Tridiagonal, B::Base.LinAlg.AbstractTriangular) in Base.LinAlg at linalg/special
+(A::Base.LinAlg.AbstractTriangular, B::Bidiagonal) in Base.LinAlg at linalg/special.j
+(A::Bidiagonal, B::Base.LinAlg.AbstractTriangular) in Base.LinAlg at linalg/special.j
+(A::Base.LinAlg.AbstractTriangular, B::Array{T,2} where T) in Base.LinAlg at linalg/s
+(A::Array{T,2} where T, B::Base.LinAlg.AbstractTriangular) in Base.LinAlg at linalg/s
+(Y::Union{Base.ReshapedArray{#s267,N,A,MI} where MI<:Tuple{Vararg{Base.Multiplicative
+(X::Union{Base.ReshapedArray{#s266,N,A,MI} where MI<:Tuple{Vararg{Base.Multiplicative
+(x::Union{Base.ReshapedArray{#s268,N,A,MI} where MI<:Tuple{Vararg{Base.Multiplicative
+(r::Range{#s268} where #s268<:Base.Dates.TimeType, x::Base.Dates.Period) in Base.Date
+(A::SparseMatrixCSC, B::SparseMatrixCSC) in Base.SparseArrays at sparse/sparsematrix
+(A::SparseMatrixCSC, B::Array) in Base.SparseArrays at sparse/sparsematrix.jl:1540
+(x::AbstractSparseArray{Tv,Ti,1} where Ti where Tv, y::AbstractSparseArray{Tv,Ti,1} w
+(x::AbstractSparseArray{Tv,Ti,1} where Ti where Tv, y::Union{Base.ReshapedArray{T,1,A
+(x::AbstractArray{#s45,N} where N where #s45<:Number) in Base at abstractarraymath.jl
+(A::AbstractArray, B::AbstractArray) in Base at arraymath.jl:38
+(A::Number, B::AbstractArray) in Base at arraymath.jl:45
+(A::AbstractArray, B::Number) in Base at arraymath.jl:48
+(index1::CartesianIndex{N}, index2::CartesianIndex{N}) where N in Base.IteratorsMD at
+(index::CartesianIndex{N}, i::Integer) where N in Base.IteratorsMD at multidimensiona
+(J1::UniformScaling, J2::UniformScaling) in Base.LinAlg at linalg/uniformscaling.jl:5
+(J::UniformScaling, B::BitArray{2}) in Base.LinAlg at linalg/uniformscaling.jl:60
+(J::UniformScaling, A::AbstractArray{T,2} where T) in Base.LinAlg at linalg/uniformsc
+(a::Base.Pkg.Resolve.VersionWeights.HierarchicalValue{T}, b::Base.Pkg.Resolve.Version
+(x::P, y::P) where P<:Base.Dates.Period in Base.Dates at dates/periods.jl:70
+(x::Base.Dates.Period, y::Base.Dates.Period) in Base.Dates at dates/periods.jl:346
+(y::Base.Dates.Period, x::Base.Dates.CompoundPeriod) in Base.Dates at dates/periods.j
+(x::Union{Base.Dates.CompoundPeriod, Base.Dates.Period}) in Base.Dates at dates/perio
+(x::Union{Base.Dates.CompoundPeriod, Base.Dates.Period}, Y::Union{Base.ReshapedArray{
+(x::Base.Dates.TimeType) in Base.Dates at dates/arithmetic.jl:8
+(a::Base.Dates.TimeType, b::Base.Dates.Period, c::Base.Dates.Period) in Base.Dates at
+(a::Base.Dates.TimeType, b::Base.Dates.Period, c::Base.Dates.Period, d::Base.Dates.Pe
+(x::Base.Dates.TimeType, y::Base.Dates.CompoundPeriod) in Base.Dates at dates/periods
+(x::Base.Dates.Instant) in Base.Dates at dates/arithmetic.jl:4
+(y::Base.Dates.Period, x::Base.Dates.TimeType) in Base.Dates at dates/arithmetic.jl:8
+(x::AbstractArray{#s268,N} where N where #s268<:Base.Dates.TimeType, y::Union{Base.Da
+(x::Base.Dates.Period, r::Range{#s268} where #s268<:Base.Dates.TimeType) in Base.Date
+(y::Union{Base.Dates.CompoundPeriod, Base.Dates.Period}, x::AbstractArray{#s268,N} wh
+(y::Base.Dates.TimeType, x::Union{Base.ReshapedArray{#s268,N,A,MI} where MI<:Tuple{Va
+(J::UniformScaling, x::Number) in Base at deprecated.jl:56
+(x::Number, J::UniformScaling) in Base at deprecated.jl:56
+(a, b, c, xs...) in Base at operators.jl:424
```

In [23]:
```
#Podemos definir mas metodos. Para esto primero tenemos que importar + de Base
import Base: +
```

In [20]:
```
#Por ejemplo si queremos concatenar elementos con +. Sin extender el metodo, no funci
```

```
         # (No esta entre esos 180 anteriores)
         +(x::String, y::String) = string(x, y)
```

```
Out[20]: + (generic function with 181 methods)
```

```
In [21]: "Hello" + " world!"
```

```
Out[21]: "Helloworld!"
```

```
In [22]: foo(x, y) = println("duck-typed foo!")
         foo(x::Int, y::Float64) = println("foo con entero y flotante!")
         foo(x::Float64, y::Float64) = println("foo con dos flotantes!")
         foo(x::Int, y::Int) = println("foo con dos enteros")
```

```
Out[22]: foo (generic function with 4 methods)
```

```
In [23]: foo(1, 1)
```

```
foo con dos enteros
```

```
In [24]: foo(1. , 1.)
```

```
foo con dos flotantes!
```

```
In [25]: foo(1, 1.0)
```

```
foo con entero y flotante!
```

```
In [26]: foo(true, false)
```

```
duck-typed foo!
```

2.1 Paradigmas que soporta

2.1.1 Funcional

```
In [17]: map(x -> x^3, (1:5))
```

```
Out[17]: 5-element Array{Int64,1}:
            1
            8
           27
           64
          125
```

```
In [5]: filter(isprime, (1:50))
```

```
Out[5]: 15-element Array{Int64,1}:
          2
          3
          5
          7
         11
         13
         17
         19
         23
         29
         31
         37
         41
         43
         47
```

2.1.2 Orientado a objetos

```
In [24]: type Tigre
           largo_de_cola::Float64
           color_de_pelaje
         end
```

```
In [19]: abstract Felino # Declaramos una clase abstracta sin comportamiento
```

```
In [20]: type Pantera <: Felino
             color_de_ojos
             color_de_pelaje
           Pantera() = new("verde", "negro")
         end
```

```
In [21]: type Leon <: Felino
           color_de_melena
           roar::AbstractString
         end
```

```
In [22]: tigger = Tigre(3.5,"naranja")
```

```
Out[22]: Tigre(3.5,"naranja")
```

```
In [23]: function meow(animal::Leon)
           animal.roar
         end

         function meow(animal::Pantera)
           "grrr"
         end
```

```
         function meow(animal::Tigre)
           "rawwwr"
         end
```

```
Out[23]: meow (generic function with 3 methods)
```

```
In [24]: meow(tigger)
```

```
Out[24]: "rawwwr"
```

```
In [25]: meow(Leon("marrón","ROAAR"))
```

```
Out[25]: "ROAAR"
```

```
In [26]: meow(Pantera())
```

```
Out[26]: "grrr"
```

3 Caracteristicas avanzadas del lenguaje

Diseñado para paralelismo y computción distribuida
Corutinas también llamadas: Lightweight threading
Macros parecidos a los de Lisp y otras facilidades para metaprogramacion
Acepta multiple dispach

4 Paralelismo

- El paralelismo en Julia se hace por CPU, no GPU.
- Los procesos se llaman Workers con un ID específico.

```
In [21]: println(nprocs())
         addprocs(1)
         println(nprocs())
```

```
1
2
```

```
In [1]: for w in workers()
            rref=remotecall(myid, w)
            sleep(1)
            println(fetch(rref))
        end
```

```
1
```

```
In [29]: @everywhere function random_num(n)
             c::Int = 0
             for i = 1:n
                 c += rand(Bool)
             end
             c
         end

         a = @spawn random_num(100000000)

         b = @spawn random_num(10000000)

         println(fetch(a)+fetch(b)) #reducción
```

```
55002896
```

```
In [33]: # Se genera la lista de tasks
         t = @task Any[ for x in [1,2,4] println(x) end ]

         # No hay tasks?
         println(istaskdone(t))

         # Cual es la siguiente task?
         println(current_task())

         # Ejecutar task
         println(consume(t))
```

```
false
Task (runnable) @0x00007fb443677490
1
2
4
Any[nothing]
```

```
In [ ]: c1 = Channel(32)
        put!(c1,3)
        put!(c1,4)
        c2 = Channel(32)

        # foo() lee un item de c1, lo imprime y lo escribe en c2
        function foo()
            while true
                data = take!(c1)
                println(data)
                result = data + 2
                put!(c2, result)
```

```
    end
end

# con @schedule podemos hacer que varias instancias de foo() estén activas concurrente
for i in 1:3
    @schedule foo()
end

for i in 1:3
    data= take!(c2)
    println(data)
end
```

```
3
4
5
6
```

```
In [1]: c = Channel(0)

        task = @schedule foreach(i->put!(c, i), 1:4)

        bind(c,task)

        for i in c
            @show i
        end

        isopen(c)
```

```
i = 1
i = 2
i = 3
i = 4
```

```
Out[1]: false
```

4.0.1 Machine Learning

KMeans: se inicializan k-centroides y se asigna a cada punto del set de datos el centroide más cercano. Se recalculan los centroides como promedio de todos los puntos y se repite hasta lograr la convergencia.

```
In [25]: using RDatasets
         xclara = dataset("cluster", "xclara")
         size(xclara)
```

```
Out[25]: (3000,2)
```

In [26]:
```
x = xclara[:V1]
y = xclara[:V2]
using Plots
scatter(x, y ,alpha=0.5)
```

```
[Plots.jl] Initializing backend: pyplot


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

In [27]:
```
using Clustering

xclara = convert(Array, xclara)
xclara = xclara'
xclara_kmeans = kmeans(xclara, 3)
```

```
sys:1: MatplotlibDeprecationWarning: The set_axis_bgcolor function was deprecated in version 2
```

Out[27]:
```
Clustering.KmeansResult{Float64}(2x3 Array{Float64,2}:
  69.9242  40.6836   9.47805
 -10.1196  59.7159  10.6861 ,[3,3,3,3,3,3,3,3,3,3    1,1,1,1,1,1,1,1,1,1],[248.826,97
```

DBSCAN: otro algoritmo de clustering. Este no recibe la cantidad de clusters como hiperparámetro y tiene la capacidad de manejar mejor los puntos entre clusters

In [28]:
```
using Distances
dclara = pairwise(SqEuclidean(), xclara);

# parametros: matriz de distancias, el radio de un cluster,
# mínimo número de puntos que forman un cluster
xclara_dbscan = dbscan(dclara, 10, 40);

# devuelve cantidad de puntos para cada cluster encontrado
xclara_dbscan.counts
```

Out[28]:
```
3-element Array{Int64,1}:
 274
 603
 356
```

5 Estadisticas

Index TIOBE Mayo 2018

Populridad en Githut en 2018

Popularidad de lenguajes en ofertas laborales en cuanto a machine learning o data science en 2016

Pero cuando cambiamos a la popularidad relativa.

6 Comparacion otros lenguajes

Julia vs Python

Ventajas de Juila:

- Rápido por default.
- Sintaxis amigable para matemáticas.
- No se pierde el manejo automatico de memoria.
- Paralelismo.

Ventajas de Python:

- Los arrays de Julia son indexados a partir del 1.
- Julia todavía es un lenguaje muy nuevo.
- Python tiene más paquetes creados para el lenguaje.
- Python tiene una comunidad más grande.

Comparación con Python: cálculo de la traza de una matriz de 10mil x 10mil

In [29]:
```
using Distributions
m = 10000
matrix = rand(Uniform(0.0, 10.0), m, m)
```

Out[29]:
```
10000x10000 Array{Float64,2}:
 5.10301   4.9494    8.4315    1.78559     1.48577   2.0306    3.7455
 7.45261   4.52143   7.02517   3.5295       9.22598   5.59536   0.0648135
 0.694313  8.77365   6.92929   5.49732      7.05565   7.33919   0.230464
 5.16956   4.06135   5.92671   6.51421      6.40954   5.51859   5.59613
 3.32278   5.83048   1.31933   1.41544      9.79483   4.85082   4.02339
 3.1647    8.10104   5.92722   4.42384     8.09922   7.89291   5.5679
 4.94211   3.96496   3.61823   2.04495      2.68936   9.36194   7.02224
 2.46266   2.02744   4.32959   0.754458     1.70075   8.47989   6.40496
 8.72047   8.06632   5.18571   6.31096      9.09257   8.86067   3.92422
 2.40688   5.79315   0.967676  6.80719      8.45674   8.62953   3.87599
 0.419425  9.56803   3.43729   7.71734     7.77613   3.06545   5.75549
 3.78544   2.1246    0.656562  9.83877      3.63004   2.03278   3.11291
 4.34213   8.61025   1.2394    1.93805      0.298725  6.23973   8.51245

 9.48088   3.46134   3.7969    2.99006      7.20339   6.69758   2.41433
 8.36319   8.16223   9.50436   2.65054      7.22037   0.242126  8.5887
 4.85172   2.75168   4.32459   3.62131     9.22106   7.60382   6.62872
 6.39179   9.55463   9.90338   6.93428      1.22896   5.35197   3.5354
 5.72619   1.99794   6.54228   6.65758      3.38225   7.40313   1.58497
 2.23976   5.87572   6.40674   7.80678      7.65019   5.49327   6.65028
 5.32705   5.38361   0.16454   2.20856      7.51542   7.43437   8.78094
```

```
 5.54444   0.349194  3.45422   9.75574     1.95739   3.99331   0.571166
 5.49523   1.71348   8.52269   7.47922      1.44497   3.42216   5.32601
 2.92527   4.12466   3.91028   9.1979       0.158091  4.42051   0.310866
 2.51086   4.39179   7.11106   1.72917      5.60529   5.55573   4.07721
 8.36307   4.73922   7.75184   3.23269      1.87024   8.0627    3.11056
```

```
In [31]: n = 0;
         for i in [1:100]
             tic()
             trace(matrix)
             aux = toq()
             n = n + aux
         end

         n/100

WARNING: [a] concatenation is deprecated; use collect(a) instead
 in depwarn at ./deprecated.jl:73
while loading In[31], in expression starting on line 2


Out[31]: 0.00015749544999999997
```

Python:

7 Casos de estudios

En 2015, el banco de reserva federal de nueva york uso julia para hacer modelos de la economía de los estados unidos.

Notaron que el modelo hecho con el lenguaje era 10 veces más rapido que el anterior (hecho con MATLAB) dado por:

- Un sistema flexible y potente que proporciona una forma natural de estructurar y simplificar la base de código.
- Multiple dispatch, lo que les permitió escribir un código más genérico.
- Un potente compilador que aumentó el rendimiento.

El proyecto Celste que cataloga estrellas y galaxias en el Apache Point Observatory de Nevo Mexico logró un gran numero de hitos:

- Logró rendimiento pico de 1.54 petaFLOPS usando 1.3 millones de threads.
- Agregó ~178 terabytes de datos en imagenes.
- Produjo parametros estimados para 188 millones de estrellas y galaxias en 14.6 minutos.
- Proporcionó no solo estimaciones para las fuentes de luz sino que, por primera vez proporcionó una medida de principios de la calidad de la inferencia para cada fuente de luz.

8 Conclusion

- Es un lenguaje nuevo pero está en crecimiento.
- Estadísticamente tiene un alto rendimiento.
- Todavía tiene una comunidad muy pequeña.

9 Referencias

https://julialang.org/
https://juliacomputing.com
https://github.com/DataWookie/MonthOfJulia
https://docs.julialang.org
https://juliabox.com/#