JULIA

TREBALL DIRIGIT

LLENGUATGES DE PROGRAMACIÓ

Grau en Enginyeria Informàtica

#12125

Universitat Politècnica de Catalunya

Maig, 2022

ÍNDEX

INTRODUCCIÓ

El llenguatge de programació Julia, és un llenguatge de programació d'alt nivell, adreçat a l'anàlisi numèrica. Especialment, s'usa per a la construcció de models matemàtics i tècniques numèriques per resoldre problemes científics, de ciències socials i problemes d'enginyeria.

Malgrat ser un llenguatge que destaca especialment en el camp de la computació científica[1], també se'l pot considerar com un llenguatge de propòsit general[2].

Julia és considerat com un llenguatge flexible, potent, fàcil d'aprendre i utilitzar i ràpid.

HISTÒRIA I ORÍGENS DE LP

Julia fou creat originalment l'any 2012, de la mà de Jeff Bezanson, Stefan Karpinski, Viral B. Shah i Alan Edelman. És un llenguatge de programació *open source*, és a dir, gratuït i en el qual tothom pot contribuir-hi.[3]

PROPÒSIT DEL LLENGUATGE

Tal com destaquen els seus creadors a la primera entrada del seu blog[4], Julia va néixer de la cobdícia de fer un llenguatge sublim, és a dir, es va crear a partir de l'ambició de voler fer un llenguatge meravellós i poderós que tingués tot allò de bo que tenien els llenguatges que feien servir els seus creadors en el seu dia a dia. Aquests eren molt pretensiosos, ja que volien que fos "***good at everything".***

Fer un llenguatge d'aquestes dimensions, i que complís amb tot el que els seus creadors necessitaven, no era una tasca senzilla. Crear un llenguatge que tingués els punts forts de cadascun d'ells sense crear-ne un que conservés els seus punts febles de cada llenguatge no era fàcil.

Per tant, buscar el millor trade-off en aquest aspecte, no era senzill d'aconseguir, però els creadors, experts en informàtica científica, aprenentatge automàtic, mineria de dades, àlgebra lineal a gran escala, computació distribuïda i paral·lela estaven entusiasmats per trobar un llenguatge que complís amb la majoria dels seus requeriments.

---

[1] La computació científica és en essència l'aplicació de la informàtica per a la resolució de problemes científics. Aquests problemes científics abasten diverses disciplines de la ciència, com ara la química, la física, les matemàtiques i la biologia o les finances, sempre que el problema es pugui modelar matemàticament.

[2] Un llenguatge de programació de propòsit general és un llenguatge dissenyat per a escriure programari en una gran varietat de camps d'aplicació diferents, és a dir per a una àmplia gamma de propòsits. Per exemple: Python, Tcl, C++. Tot i que cal notar, que cada llenguatge té un camp o domini en què és el millor, Python, per exemple és millor com a llenguatge de scripting.

Altrament, un llenguatge més específic, o que satisfà una necessitat particular, s'anomena llenguatge de programació específic de domini, ja que s'usa per satisfer les necessitats d'un entorn en concret. Per exemple: HTML o bé SQL.

[3] Pots contribuir a l'ampliació i millora del llenguatge en el següent repositori de GitHub: https://github.com/JuliaLang

[4] https://julialang.org/blog/2012/02/why-we-created-julia/

RELACIÓ AMB LPs SEMBLANTS

Julia és un llenguatge de programació que destaca especialment per la seva facilitat i expressivitat a l'hora de tractar amb problemes numèrics d'alt nivell. Aquesta característica és també present en llenguatges com: R, MATLAB i Python.

Tanmateix, la seva relació amb altres LPs no conclou aquí, Julia és un llenguatge *open source*, una característica que tenen llenguates coneguts com: JavaScript, Python, PHP, R, Go, Scala, Kotlin i Ruby, entre molts altres.

La seva relació amb LPs semblants es pot resumir com segueix a continuació:

- Potent i amb la velocitat de C
- Aprofita el dinamisme de Ruby i Lua.
- És un llenguatge homoicònic, amb macros reals com Lisp
- Usable per a la programació general com Python
- Fàcil per a manegar les estadístiques com R
- Senzill alhora de processar strings com ho fa Perl
- Tan potent per a l'àlgebra lineal i amb una notació matemàtica com MATLAB
- Fàcil d'aprendre com Python, i finalment
- Interpretat, com PHP o Python, i també possible de compilar com C

D'altra banda, Julia també és un llenguatge de programació que és una alternativa al llenguatge Hadoop[5]. Julia ens permet de forma senzilla, dividir un problema més gran en molts problemes més petits, i distribuir-los entre molts sistemes tot aplicant el principi de "paral·lelisme del disseny".

PARADIGMES DE PROGRAMACIÓ QUE ADMET EL LLENGUATGE

Julia és un llenguatge de programació multiparadigma. Agafa referències imperatives, funcionals, de *multiple dispatch* i meta.

IMPERATIVA

Podrem precisar a través de les instruccions el canvi d'estat en el nostre programa. Per fer-ho, per exemple, podrem fer ús dels operadors d'assignació "=" i de suma "+=" per poder canviar el contingut de les variables en memòria.

[5] Hadoop és un *framework open source* que s'utilitza per emmagatzemar i processar de manera eficient grans conjunts de dades que van des de gigabytes fins a petabytes de dades. En lloc de fer servir un ordinador gran per emmagatzemar i processar les dades, Hadoop permet agrupar diversos ordinadors per analitzar conjunts de dades massius en paral·lel més ràpidament.

FUNCIONAL

Julia ens ofereix característiques que estan presents en llenguatges de programació funcionals, però aquest no és el seu principal paradigma de programació. Julia, no ens pot oferir moltes coses que ofereixen la majoria de llenguatges funcionals, com són:

- Composició de funcions i
- l'aplicació parcial de funcions malgrat disposar de funcions d'ordre superior
- Les estructures de dades a Julia són mutables (en la majoria de llenguatges purament funcionals solen treballar amb estructures de dades immutables per defecte),
- a causa de la mutabilitat, tindrem operadors com "==", "+=", "++", fet que en un llenguatge de programació funcional, no en disposem.

MULTIPLE DISPATCH

El principal paradigma de Julia és el de *multiple dispatch*. I és que aquesta característica del llenguatge ens permet no haver de tenir una crida separada per a cada tipus de dades, si aquestes funcions fan el mateix.

```
julia> describe(n::Integer) = "integer $n"
describe (generic function with 1 method)

julia> describe(n::AbstractFloat) = "floating point $n"
describe (generic function with 2 methods)
```

Per tant, no caldrà especificar de quin tipus és l'argument que estem a punt de passar dins d'una crida del mètode.

```
julia> describe(10)
"integer 10"

julia> describe(1.0)
"floating point 1.0"
```

També podem crear mètodes per a més d'un argument. Això és útil quan es defineixen mètodes especialitzats per a operacions en determinats tipus i mètodes alternatius per a altres tipus.

```
julia> describe(n::Integer, m::Integer) = "integers n=$n and m=$m"
describe (generic function with 3 methods)

julia> describe(n, m::Integer) = "only m=$m is an integer"
describe (generic function with 4 methods)

julia> describe(n::Integer, m) = "only n=$n is an integer"
describe (generic function with 5 methods)
```

D'igual forma, a la crida del mètode no caldrà especificar de quin tipus és l'argument que estem a punt de passar.

```
julia> describe(10, 'x')
"only n=10 is an integer"

julia> describe('x', 10)
"only m=10 is an integer"

julia> describe(10, 10)
"integers n=10 and m=10"
```

META

Lisp, un dels llenguatges que ha influenciat Julia, ha permès que aquest doni suport a la metaprogramació. Igual que Lisp, Julia representa el seu propi codi com una estructura de dades del mateix llenguatge. Com que el codi està representat per objectes que es poden crear i manipular des del llenguatge, és possible que un programa transformi i generi el seu propi codi o fins i tot modificar-se mentre s'executa.

SISTEMA D'EXECUCIÓ

Julia és un llenguatge de programació que compta amb un sistema d'execució mixt. És a dir, un sistema de compilació dinàmica anomenat JIT (*Just in Time Compilation*). Aquest JIT, en el cas de Julia, s'ha implementat amb LLVM[6].

La idea darrere de la compilació JIT és aportar els avantatges de la compilació estàtica als programes interpretats per poder millorar-ne el rendiment.

Pel que fa a aquest sistema, ens permetrà mentre s'executa el programa interpretat, determinar el codi més utilitzat, per posteriorment compilar-lo a codi màquina.

Durant aquest escenari JIT, el sistema prendrà el codi font i el convertirà, en el cas de Julia, a un llenguatge intermedi. Aquest codi intermedi, posteriorment es convertirà en llenguatge màquina.

Tot i això, cal notar que a JIT, no tot el codi intermedi es convertirà a codi de màquina. Per poder discernir què es traduirà a codi màquina i què no necessitarem mantenir un recompte de quantes vegades s'executa una funció o una certa secció de codi. Si aquest recompte supera un límit predefinit, JIT compilarà el codi en llenguatge màquina per posteriorment ser executat directament pel processador.

També la pròxima vegada que es calculi aquesta funció, s'executarà de nou el mateix codi compilat a diferència de la interpretació normal en què el codi s'interpreta de nou línia per línia, fent que l'execució sigui més ràpida amb aquest tipus de compilació JIT.

Un exemple de compilador JIT és la màquina virtual de Java, JVM.

SISTEMA DE TIPUS

Julia és un llenguatge fortament tipat i, per tant, ens ajudarà a evitar cometre errors si intentem barrejar valors de diferents tipus.

A més a més, és un llenguatge d'assignació dinàmica i, per tant, no caldrà que indiquem el tipus de les nostres variables, característica, per exemple, que comparteix amb Python.

[6] LLVM és una infraestructura de compilació que a través d'un conjunt d'eines ens permet crear JITs per a nous llenguatges informàtics gràcies a les comandes bàsiques que proporciona.

Tot i això, opcionalment podem decidir, assignar un tipus a les nostres variables i expressions. Aquesta assignació de tipus, en certes ocasions, permetrà al nostre JIT inferir millors optimitzacions a l'hora de compilar el codi.

PRINCIPALS APLICACIONS

Malgrat ser un llenguatge de programació relativament nou, en l'actualitat, molts productes desenvolupats amb Julia ja s'estan aplicant en nombroses indústries en mans de: *Software Engineers*, *Data Scientists* passant per comerciants, farmacèutics, banquers i gestors de TI.

Un dels exemples de productes més establerts en nombroses empreses són:

- JuliaSim[7] i
- JuliaHub[8]

Entre algunes indústries que disposen d'aquests productes, estarien:

- Farmacèutica
- Financera
- Mèdica
- Energètica
- Governamental

Aquestes indústries necessiten resoldre problemes específics del seu camp, on cerquen amb els productes que ofereix Julia, millores de rendiment significatives, buscant:

- una escalabilitat massiva,
- un augment de la productivitat
- i a la vegada una disminució del temps de llançament al mercat del seu producte.

A continuació, en podeu veure un tastet d'empreses que fan servir els productes de Julia:

[7] És una plataforma de simulació al núvol, que combina les últimes tècniques basades en Machine Learning que acceleren la simulació fins a 500x.
[8] Permet desenvolupar aplicacions amb un IDE al navegador allotjat al núvol. JuliaHub ofereix als científics, enginyers i innovadors tota la potència informàtica d'alt rendiment que necessiten per realitzar idees innovadores, a qualsevol escala que vulguin.

EXEMPLES DE CODI IL·LUSTRATIUS

A continuació podeu veure un seguit d'exemples il·lustratius de la sintaxi de Julia. Per poder comparar la seva sintaxi, he cregut que pot ser rellevant comparar-lo amb un LP semblant, com és Python.

Exemples: Julia Python

- Com printar?

```julia
println("Hola! Sóc la hash: #12125")
```

```python
print("Hola! Sóc la hash: #12125")
```

- Com comentar?

```julia
# Això és un comentari

#=
Això és un comentari en
dues línies
=#
```

```python
# Això és un comentari

"""
Això és un comentari en
dues línies
"""
```

- Com crear un diccionari?

```julia
dict = Dict("1" => "Q4", "2" => "Q5",
"3" => ("Q6" => "LP",  "Q7" => "AA", "Q8" => "TFG"))

println(dict)
```

```python
Dict = {1: 'Q4', 2: 'Q5',
       3:{'Q6' : 'LP', 'Q7' : 'AA', 'Q8' : 'TFG'}}

print(Dict)
```

- Com crear i emplenar una matriu?

```julia
m, n = 5, 5
A = fill(0, (m, n))

for j in 1:n
    for i in 1:m
        A[i, j] = i + j
    end
end
A
```

```python
!pip install numpy

import numpy as np

m, n = 5, 5
A = np.zeros((n,m))

for j in range (1,n):
    for i in range (1,m):
        A[i, j] = i + j
```

- Com fer una cerca binària recursiva?

```julia
function binarysearch(lst::Vector{T}, value::T, low=1, high=length(lst)) where T
    if isempty(lst) return 0 end
    if low ≥ high
        if low > high || lst[low] != value
            return 0
        else
            return low
        end
    end
    mid = (low + high) ÷ 2
    if lst[mid] > value
        return binarysearch(lst, value, low, mid-1)
    elseif lst[mid] < value
        return binarysearch(lst, value, mid+1, high)
    else
        return mid
    end
end
```

```python
def binary_search_recursive(a_list, item):
    first = 0
    last = len(a_list) - 1

    if len(a_list) == 0:
        return '{item} was not found in the list'.format(item=item)
    else:
        i = (first + last) // 2
        if item == a_list[i]:
            return '{item} found'.format(item=item)
        else:
            if a_list[i] < item:
                return binary_search_recursive(a_list[i+1:], item)
            else:
                return binary_search_recursive(a_list[:i], item)
```

ALTRES CARACTERÍSTIQUES PARTICULARS

PARAL·LELISME I PROGRAMACIÓ CONCURRENT

Julia admet quatre categories de programació concurrent i paral·lela, proporcionant-nos un entorn de multiprocessament que permet que els programes s'executin en múltiples processadors de memòria compartida o distribuïda. A continuació en podeu veure tres d'elles:

- **Tasques asíncrones**: Julia ens permet programar tasques en diversos *threads*.

- ***Multithreading:*** Julia ofereix la possibilitat de programar tasques simultàniament en més d'un fil o nucli de CPU, compartint-ne la memòria.
  Aquesta sol ser la manera més senzilla d'aconseguir o bé paral·lelisme al nostre ordinador o en un HPC.

  Per indicar-li a Julia que faci servir un nombre n de threads: podem llançar la comanda:

  ```
  julia -t n
  ```

  Seguidament podrem fer:

  ```
  julia -t n Threads.@threads for N = 1::20
        println("N = $N (thread $(Threads.threadid()) of out $(Threads.nthreads()))")
  end
  ```

- **Computació distribuïda:** La computació distribuïda executa diversos processos de Julia amb espais de memòria separats. Aquests poden estar al mateix ordinador o en diversos ordinadors.
  La biblioteca estàndard distribuïda proporciona la capacitat per a l'execució remota d'una funció Julia.

  Amb aquest bloc bàsic, és possible construir molts tipus diferents d'abstraccions de computació distribuïda. Paquets com `DistributedArrays.jl` són un exemple d'aquesta abstracció. D'altra banda, paquets com `MPI.jl` i `Elemental.jl` proporcionen accés a l'ecosistema de biblioteques MPI existent.

  Podem fer un bucle for que executant-se de forma paral·lela, de la següent manera:

  ```
  using Distributed
  @distributed for N in 1:5:20
      println("The N of this iteration in $N")
  end
  ```

  Cal notar que cada procés té les seves pròpies variables per defecte.

POPULARITAT DEL LP ELS DARRERES 10 ANYS

Aquesta secció tracta la popularitat del llenguatge de programació Julia al llarg d'aquests darrers 10 anys des de la seva creació (2013-2022). Aquesta popularitat és mesurada amb l'índex de TIOBE[9].

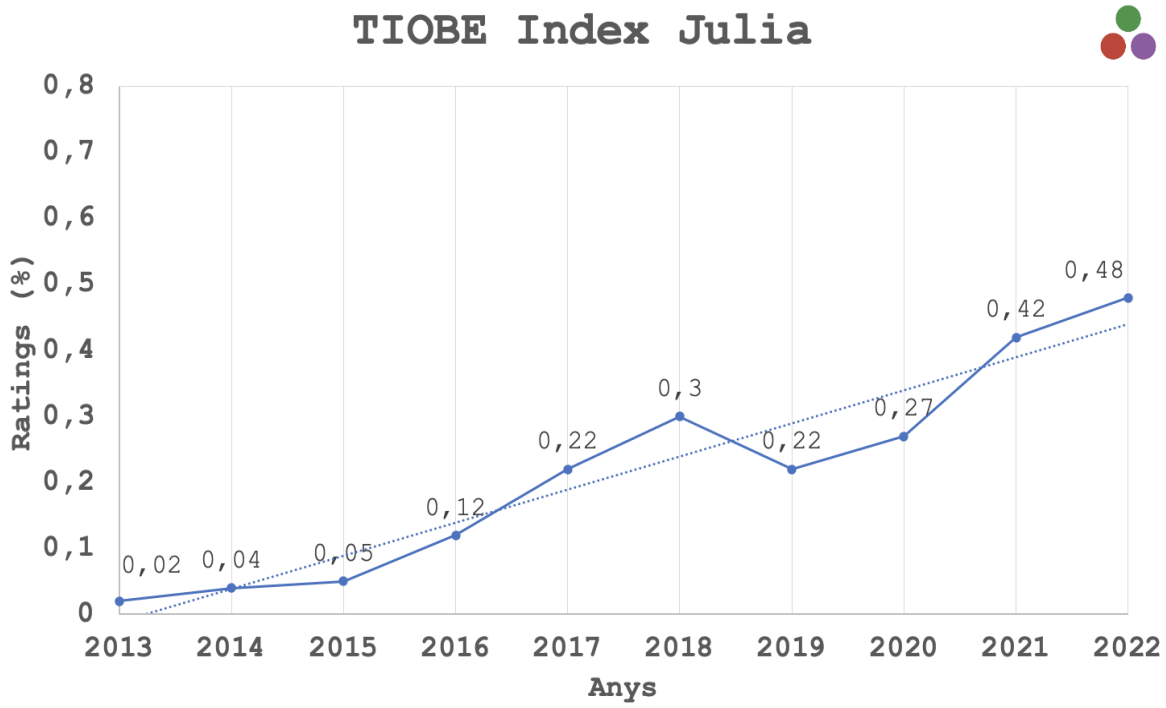
L'índex que ens mostra aquesta rellevància del llenguatge s'actualitza cada mes i ens mostra amb un valor en forma de percentatge, la popularitat que se li atribueix. TIOBE usa 25 motors de cerca per poder calcular aquest valor de la popularitat o com l'anomenarem a partir d'ara, "Rating". És clar, que aquests 25 motors de cerca no són escollits a l'atzar, sinó que tots ells comprenen uns certs requeriments.

Aquesta puntuació que hem anomenat "Rating", es veu incrementada cada cop que en alguns dels 25 motors de cerca se'n fa una query relacionada amb el llenguatge, és a dir, una visita a una pàgina fent servir la consulta de cerca de la forma: [algun mot més] + "Julia programming" + [algun mot més].

Per tant, el nombre de visites emprant la query: [algun mot més] + "Julia programming" + [algun mot més], per cada motor de cerca, contribueix a determinar el "Rating" d'un llenguatge. És clar que aquest "Rating" es normalitza per a cada motor de cerca per a tots els llenguatges de programació, que junts conformen el 100% del "Rating" possible.

$$Rating\ Julia\ =\ (visites(Julia, motorC1)/visites(motorC1)\ +\ldots+\ visites(Julia, motorCn)/visites(motorCn))/n$$

Per poder obtenir aquest "Rating" anual de Julia he fet ús de l'API de Wayback Machine, que m'ha permès consultar per cada any, pels mesos de Abril-Maig, el "Rating" obtingut per aquell mes de l'any



[9] TIOBE és un índex que mesura la popularitat dels llenguatges de programació. Creat per la mateixa companyia que rep nom aquest índex i amb origen als Països Baixos.

Tal com es pot veure a la gràfica, Julia no podria considerar-se al llarg d'aquests darrers 10 anys, com un dels llenguatges de programació més populars de tots. Aquesta popularitat, potser cal notar que no es refereix al llenguatge "preferit" o bé al que té més línies de codi, però si bé aquest l'índex ens permet inferir d'alguna forma la quantitat de persones, treballs, cursos o bé recursos, relacionats amb aquest llenguatge de programació.

Tot i això, és important destacar l'auge de popularitat des de l'any 2019. Però és clar, que el fet que sigui un llenguatge relativament nou dificulta que es trobi en posicions altes del ranking TIOBE. A més a més, actualment un llenguatge molt potent i consolidat en l'àmbit de ML i ciència de dades, és Python i en aquest sentit, és difícil d'equiparar-lo en termes de popularitat.

VISIÓ PERSONAL I CRÍTICA DEL LP

Personalment, m'ha agradat molt descobrir aquest nou llenguatge. Crec que és molt potent i en un futur no gaire llunyà podria ser un llenguatge més usat en el dia a dia de molts programadors.

Tot i això, cal tenir una visió realista i com és sabut, avui dia, a causa del rellevant auge de la intel·ligència artificial i l'aprenentatge automàtic, a hores d'ara Python és el llenguatge més comunament estès entre la comunitat de científics de dades. Per tant, en aquest camp, és difícil que agafi embranzida ara com ara.

Per altra banda, en la computació paral·lela i en la de HPC sí que hi veig un potencial més imminent, ja que moltes infraestructures actuals que es troben al núvol, ja fan servir Julia com a principal llenguatge de programació.

També m'agradaria destacar, que realment és un llenguatge senzill d'aprendre en termes de sintaxi. Si alguna vegada has fet servir MATLAB o Python t'hi sentiràs comode, i a més a més, veuràs que és un llenguatge molt ràpid si tens en consideració les optimitzacions que s'hi poden fer, i per tant, si ets molt fan de C i de Python, trobaràs a Julia el llenguatge de busques.

Com a crítica, cal dir que m'he enganxat un parell de vegades amb les matrius de Julia, ja que estan indexades en 1, cosa que us pot anar una mica malament de vegades si esteu acostumats a Python, C/++, Java, etc. I també destacar que a les matrius, Julia hi accedim per ordre de columnes, mentre que a les matrius Python Numpy s'hi accedeix per ordre de files.
Això podria afectar algunes decisions de disseny de com iterar sobre matrius de manera eficient a la memòria.

DESCRIPCIÓ DE LES FONTS D'INFORMACIÓ EMPRADES

Tota la informació que es troba en aquest document sobre Julia ha set íntegrament extreta de la documentació del llenguatge que es troba en format web a la pàgina oficial del llenguatge. Aquest fet m'ha sorprès gratament, perquè realment l'elaboració de la documentació de Julia és altament autocontinguda i amb molts exemples de tots els conceptes teòrics i les característiques que exposen.

A més a més, la documentació és fàcil de seguir, cosa que m'ha permès no perdre el fil en moltes ocasions i pràcticament els quatre primers capítols del manual m'han permès respondre la majoria d'apartats, exceptuant-ne el cas de l'apartat de paradigmes de programació, on he recorregut al blog de la pàgina oficial de Julia, on s'explicava mitjançant un vídeo elaborat per un dels creadors

(JuliaCon 2019 | The Unreasonable Effectiveness of Multiple Dispatch | Stefan Karpinski) el principal paradigma en el qual es fonamenta Julia.

Pel que fa als exemples, han estat elaborats gràcies a exemples exposats en el manual del llenguatge i gràcies als tutorials que ofereix la pàgina del pròpi llenguatge a l'apartat de *Learn (https://julialang.org/learning/).*

Respecte a un dubte del tipage del llenguatge, he consultat un recurs en format online, però que correspon a un llibre en format paper i que es pot trobar actualment a Google Books, anomenat, **Julia: High Performance Programming**

Finalment, les dades de popularitat del llenguatge de programació, han set extretes gràcies a l'API de WayBack Machine mitjançant un script que m'ha permès saber pels mesos d'Abril-Maig la popularitat de Julia entre la comunitat de programadors.

AVALUAR LA QUALITAT DE LA INFORMACIÓ TROBADA

Des del meu punt de vista, la qualitat de la informació trobada compleix amb el requisit de la rigorositat i fiabilitat, ja que és informació és en la majoria de la seva totalitat extreta de la documentació publicada pels mateixos creadors del llenguatge, exceptuant-ne la consulta al llibre en format digital, on també aquests, consulten com a font la documentació de Julia.

D'altra banda, també puc assegurar la validesa de les dades obtingudes amb l'script, ja que per comprovar i garantir que les dades eren correctes he consultat per alguns anys, l'índex TIOBE que Google em proporcionava gràcies a les seves entrades indexades.

REFERÈNCIES BIBLIOGRÀFIQUES

Jeff Bezanson, e. (2022). *The Julia Language Blog*. Julialang.org. Retrieved 2 May 2022, from https://julialang.org/blog/.

Julia Computing - Julia Computing. Juliacomputing.com. (2022). Retrieved 2 May 2022, from https://juliacomputing.com.

Julia Documentation · The Julia Language. Docs.julialang.org. (2022). Retrieved 28 April 2022, from https://docs.julialang.org/en/v1/.

Metaprogramming · The Julia Language. Docs.julialang.org. (2022). Retrieved 28 May 2022, from https://docs.julialang.org/en/v1/manual/metaprogramming/.

Usage, G., & →, n. (2022). *Understanding multiple dispatch*. JuliaLang. Retrieved 28 April 2022, from https://discourse.julialang.org/t/understanding-multiple-dispatch/76601/4.

Wayback Machine APIs | Internet Archive. Archive.org. (2022). Retrieved 29 April 2022, from https://archive.org/help/wayback_api.php.

Julia: High Performance Programming. Google Books. (2022). Retrieved 27 April 2022, from https://books.google.es/books?id=PJ_cDgAAQBAJ&pg=PA34&lpg=PA34&dq=type+checking+julia&source=bl&ots=XDvUCibEH9&sig=ACfU3U3u1EEOstBMJa4HI9Uys6pxof_Jdg&hl=ca&sa=X&ved=2ahUKEwjawdak6cX3AhUP3BoKHR_jBWEQ6AF6BAgSEAM#v=onepage&q=type%20checking%20julia&f=false.

ANNEX

COM INSTAL·LAR JULIA

Instal·lar Julia al nostre sistema operatiu, és un procediment ben senzill, simplement cal anar a la pàgina oficial de **Julia**, concretament a l'apartat de *Downloads*, i cercar la versió descarregable més adient pel nostre S.O.
Altrament, podem decidir fer-ho nosaltres mateixos des de GitHub, amb el **codi font** de Julia seguint els passos que apareixen al fitxer `README.md` del repositori.
De la mateixa manera, podem decidir instal·lant-nos el Kernel de Julia, anomenat **IJulia**, per a Jupyter Notebook.