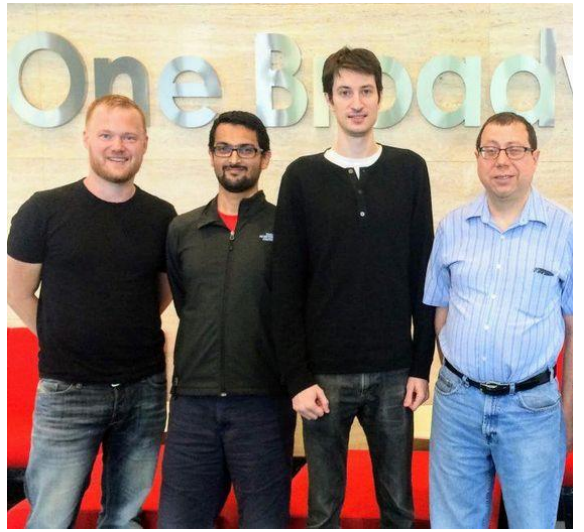
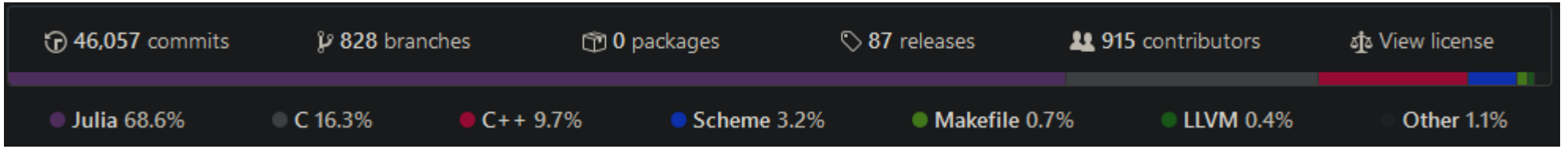
Introduction à Julia

Les débuts

- Crée en 2009 et dévoilé au public en 2012 par 4 personnes:
  - Stefan Karpinski
  - Viral B. Shah
  - Jeff Bezanson
  - Alan Edelman
- Actuellement, la version 1.4 est disponible





Les ambitions du langage 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 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 gluing 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.

Les ambitions du langage 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 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 gluing 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.

Quelques chiffres sur Julia

Cumulative Julia Growth Statistics	Total as of Jan 1, 2019	Total as of Jan 1, 2020	Growth
Number of News Articles Mentioning Julia or Julia Computing	253	468	+85%
Discourse Views (Julia Forums)	12,656,734	22,920,570	+81%
Julia Downloads (JuliaLang.org + Docker Hub + JuliaPro)	7,305,737	12,950,630	+77%
Published Citations of Julia: A Fast Dynamic Language for Technical Computing (2012) + Julia: A Fresh Approach to Numerical Computing (2017)	1,048	1,680	+60%
YouTube Julia Language Channel Views	1,013,276	1,562,223	+54%

Quelques objets de base

Entrée [1]:
```julia
# Quelques objets de base
a = 5
b = 5.
c = 5 + 2im
d = "Hello World!"
e = [3, 4, 5]
f = (3, 4, 5)
g = Set([3, 4, 4, 5])
h = Dict("key" => "value", "key2" => "value2")

Types = ("Int", "Float", "Complex", "String",
         "Array", "Tuple", "Set", "Dictionary")
for (i, j) in zip(Types, (a, b, c, d, e, f, g, h))
    println(i, ": ", j)
end
```

```
Int: 5
Float: 5.0
Complex: 5 + 2im
String: Hello World!
Array: [3, 4, 5]
Tuple: (3, 4, 5)
Set: Set([4, 3, 5])
Dictionary: Dict("key" => "value","key2" => "value2")
```

Entrée [1]:
```python
# Quelques objets de base
a = 5
b = 5.
c = 5 + 2j
d = "Hello World!"
e = [3, 4, 5]
f = (3, 4, 5)
g = set([3, 4, 4, 5])
h = {"key": "value", "key2": "value2"}

Types = ("Int", "Float", "Complex", "String",
         "Array", "Tuple", "Set", "Dictionary")
for i, j in zip(Types, [a, b, c, d, e, f, g, h]):
    print(i, ":", j)
```

```
Int : 5
Float : 5.0
Complex : (5+2j)
String : Hello World!
Array : [3, 4, 5]
Tuple : (3, 4, 5)
Set : {3, 4, 5}
Dictionary : {'key': 'value', 'key2': 'value2'}
```

Entrée [1]:
```r
# Quelques objets de base
a = 5
b = 5.
c = 5 + 2i
d = "Hello World!"
e = c(3, 4, 5)

print(paste("Int: ", a))
print(paste("Float: ", b))
print(paste("Complex: ", c))
print(paste("String: ", d))
print(paste("Array:"))
print(e)
```

```
[1] "Int:  5"
[1] "Float:  5"
[1] "Complex:  5+2i"
[1] "String:  Hello World!"
[1] "Array:"
[1] 3 4 5
```

Le typage en Julia (pour les arrays)

Entrée [2]:
```
a = [1, 2, 3]
a[1] = "Hello World!"
```

```
MethodError: Cannot `convert` an object of type Str
ing to an object of type Int64
Closest candidates are:
  convert(::Type{T<:Number}, !Matched::T<:Number) w
here T<:Number at number.jl:6
  convert(::Type{T<:Number}, !Matched::Number) wher
e T<:Number at number.jl:7
  convert(::Type{T<:Integer}, !Matched::Ptr) where
T<:Integer at pointer.jl:23
  ...

Stacktrace:
 [1] setindex!(::Array{Int64,1}, ::String, ::Int64)
at .\array.jl:766
 [2] top-level scope at In[2]:2
```

Entrée [2]:
```
a = [1, 2, 3]
a[0] = "Hello World!"
print(a)
```

```
['Hello World!', 2, 3]
```

Entrée [3]:
```
import numpy as np
array = np.array([4, 5, 6])
print(array)
```

```
[4 5 6]
```

Entrée [4]:
```
new_array = np.array([3, 4, 5])
new_array[2] = "Hello World!"
print(new_array)
```

```
---------------------------------------------------
----------------------
ValueError                                Traceback
(most recent call last)
<ipython-input-4-36bd49af4099> in <module>
      1 new_array = np.array([3, 4, 5])
----> 2 new_array[2] = "Hello World!"
      3 print(new_array)

ValueError: invalid literal for int() with base 10:
'Hello World!'
```

Entrée [2]:
```
a = c(1, 2, 3)
a[1] = "Hello World!"
print(a)
```

```
[1] "Hello World!" "2"           "3"
```

Opérations mathématiques

Entrée [4]:
```
# Addition
array = [3, 4, 5]
print(array + array)
print(array + 10)
```

```
[6, 8, 10]

MethodError: no method matching +(::Array{Int64,1}, ::Int
64)
Closest candidates are:
  +(::Any, ::Any, !Matched::Any, !Matched::Any...) at ope
rators.jl:529
  +(!Matched::Complex{Bool}, ::Real) at complex.jl:293
  +(!Matched::Missing, ::Number) at missing.jl:94
  ...

Stacktrace:
 [1] top-level scope at In[4]:4
```

Entrée [5]:
```
# Addition
array = np.array([3, 4, 5])
print(array + array)
print(array + 10)
```

```
[ 6  8 10]
[13 14 15]
```

Entrée [3]:
```
# Addition
array = c(3, 4, 5)
print(array + array)
print(array + 10)
```

```
[1]  6  8 10
[1] 13 14 15
```

Dot operator

Entrée [5]:
```
# Quelques exemples du dot operator
array = [3, 4, 5]
println(sin.(array))

println(array.^3)

println(uppercase.(["hello", "world"]))
```

```
[0.1411200080598672, -0.7568024953079282, -0.9589242746631
385]
[27, 64, 125]
["HELLO", "WORLD"]
```

Entrée [6]:
```
# Quelques exemples des différences avec le dot operator
array = np.array([3, 4, 5])
print(np.sin(array))

print(array**3)

print([i.upper() for i in ["hello", "world"]])
```

```
[ 0.14112001 -0.7568025  -0.95892427]
[ 27  64 125]
['HELLO', 'WORLD']
```

Entrée [4]:
```
# Quelques exemples des différences avec le dot operator
array = c(3, 4, 5)
sin(array)

array^3

toupper(c("hello", "world"))
```

```
0.141120008059867  -0.756802495307928  -0.958924274663138

27  64  125

'HELLO'  'WORLD'
```

Opérations mathématiques

Entrée [6]:
```
# Addition
array = [3, 4, 5]
println(array + array)
println(array .+ 10)
```

```
[6, 8, 10]
[13, 14, 15]
```

Entrée [11]:
```
# Multiplication of array
x = [3, 4, 5]

println("Multiply by 2:")
println(2x)

println("Multiply by array:")
println(x .* x)

println("Inner Product:")
println(x'x)

println("Outer Product:")
println(x * x')
```

```
Multiply by 2:
[6, 8, 10]
Multiply by array:
[9, 16, 25]
Inner Product:
50
Outer Product:
[9 12 15; 12 16 20; 15 20 25]
```

Entrée [5]:
```
# Addition
array = np.array([3, 4, 5])
print(array + array)
print(array + 10)
```

```
[ 6  8 10]
[13 14 15]
```

Entrée [7]:
```
# Multiplication of array
x = np.array([3, 4, 5])

print("Multiply by 2:")
print(2 * x)

print("Multiply by array:")
print(x * x)

print("Inner Product:")
print(np.dot(x, x))

print("Outer Product:")
print(np.outer(x, x))
```

```
Multiply by 2
[ 6  8 10]
Multiply by array
[ 9 16 25]
Inner Product: 50
Outer Product:
 [[ 9 12 15]
 [12 16 20]
 [15 20 25]]
```

Entrée [3]:
```
# Addition
array = c(3, 4, 5)
print(array + array)
print(array + 10)
```

```
[1]  6  8 10
[1] 13 14 15
```

Entrée [7]:
```
# Multiplication of array
x = c(3, 4, 5)

print("Multiply by 2")
2 * x

print("Multiply by array")
x * x

# Inner Product
print("Inner Product")
x %*% x

# Outer Product
print("Outer Product:")
outer(x, x)
```

```
[1] "Multiply by 2"
```

```
6  8  10
```

```
[1] "Multiply by array"
```

```
9  16  25
```

```
[1] "Inner Product"
```

```
50
```

```
[1] "Outer Product:"
```

9	12	15
12	16	20
15	20	25

Opérations mathématiques

Entrée [16]:
```
# Concatenate arrays
x = [3, 4, 5]

println("Verttical Concatenation:")
println([x, x])

println("Horizontal Concatenation:")
println([x x])
```

```
Verttical Concatenation:
Array{Int64,1}[[3, 4, 5], [3, 4, 5]]
Horizontal Concatenation:
[3 3; 4 4; 5 5]
```

Entrée [8]:
```
# Concatenate arrays
x = np.array([3, 4, 5])

print("Verttical Concatenation:")
print(np.vstack((x, x)))

print("Horizontal Concatenation:")
print(np.dstack((x, x)))
```

```
Verttical Concatenation:
[[3 4 5]
 [3 4 5]]
Horizontal Concatenation:
[[[3 3]
  [4 4]
  [5 5]]]
```

Entrée [11]:
```
# Concatenate arrays
x = c(3, 4, 5)
print("Verttical Concatenation:")
rbind(x, x)

print("Horizontal Concatenation:")
cbind(x, x)
```

```
[1] "Verttical Concatenation:"
```

array	3	4	5
array	3	4	5

```
[1] "Horizontal Concatenation:"
```

array	array
3	3
4	4
5	5

Composition et piping

Entrée [2]:
```
# Function composition and piping
println("sqrt(sum(1:10)) = $(sqrt(sum(1:10)))")
println("1:10 |> sum |> sqrt = $(1:10 |> sum |> sqrt)")
println("(sqrt ∘ sum)(1:10) = $((sqrt ∘ sum)(1:10))")
```

```
sqrt(sum(1:10)) = 7.416198487095663
1:10 |> sum |> sqrt = 7.416198487095663
(sqrt ∘ sum)(1:10) = 7.416198487095663
```

Entrée [22]:
```
# Function composition and piping
print(paste("sqrt(sum(1:10))", sqrt(sum(1:10))))
print(paste("1:10 %>% sum %>% sqrt:", 1:10 %>% sum %>% sqrt))
```

```
[1] "sqrt(sum(1:10)) 7.41619848709566"
[1] "1:10 %>% sum %>% sqrt: 7.41619848709566"
```

Multiple dispatch

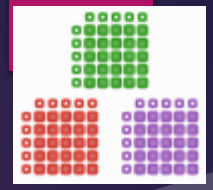
Entrée [9]:
```
# Exemple du multple dispatch
func(x::Number, y::Number) = 2x + y
func(x::String, y::String) = "$x, $y"

println(@which func(5,6))
println(func(5, 6))

println(@which func("Hello", "World"))
println(func("Hello", "World"))

println(func("Hello", 5))
```

```
func(x::Number, y::Number) in Main at In[9]:2
16
func(x::String, y::String) in Main at In[9]:3
Hello, World

MethodError: no method matching func(::String, ::Int64)
Closest candidates are:
  func(::String, !Matched::String) at In[9]:3
  func(!Matched::Number, ::Number) at In[9]:2

Stacktrace:
 [1] top-level scope at In[9]:10
```

Entrée [10]:
```
println("Methodes pour *: $(length(methods(*)))")
println("Methodes pour +: $(length(methods(+)))")
println("Methodes pour -: $(length(methods(-)))")
println("Methodes pour length: $(length(methods(length)))")
```

```
Methodes pour *: 354
Methodes pour +: 161
Methodes pour -: 168
Methodes pour length: 85
```

Lecture de données avec DataFrames



Entrée [27]:
```
using DataFrames, CSV
city = raw"C:\Users\mteissier\Downloads\toy_dataset"
city = joinpath(city, raw"toy_dataset.csv")

df = CSV.read(city)
first(df, 10)
```

Out[27]: 10 rows × 6 columns

	Number	City	Gender	Age	Income	Illness
	Int64	String	String	Int64	Float64	String
1	1	Dallas	Male	41	40367.0	No
2	2	Dallas	Male	54	45084.0	No
3	3	Dallas	Male	42	52483.0	No
4	4	Dallas	Male	40	40941.0	No
5	5	Dallas	Male	46	50289.0	No
6	6	Dallas	Female	36	50786.0	No
7	7	Dallas	Female	32	33155.0	No
8	8	Dallas	Male	39	30914.0	No
9	9	Dallas	Male	51	68667.0	No
10	10	Dallas	Female	30	50082.0	No

Entrée [28]:
```
country = raw"C:\Users\mteissier\Downloads\toy_dataset"
country = joinpath(country, "toy_country.csv")

df_country = CSV.read(country)
first(df_country, 10)
```

Out[28]: 8 rows × 2 columns

	City	Country
	String	String
1	Dallas	USA
2	New York City	USA
3	Los Angeles	USA
4	Mountain View	USA
5	Boston	USA
6	Washington D.C.	USA
7	San Diego	USA
8	Austin	USA

- Tutoriel:

https://juliadata.github.io/DataFrames.jl/stable/

Sélection des données avec Query

Entrée [13]:
```
using Query
x = @from i in df begin
    @join j in df_country on i.City equals j.City
    @orderby descending(i.Age), i.City, j.Country
    @where i.Age > 50
    @select {i.City, i.Age, i.Illness, j.Country}
    @collect DataFrame
end
first(x, 10)
```

Out[13]: 10 rows × 4 columns

	City	Age	Illness	Country
	String	Int64	String	String
1	Austin	65	No	USA
2	Austin	65	No	USA
3	Austin	65	No	USA
4	Austin	65	No	USA
5	Austin	65	No	USA
6	Austin	65	No	USA
7	Austin	65	No	USA
8	Austin	65	No	USA
9	Austin	65	No	USA
10	Austin	65	No	USA

- Plusieurs syntaxes sont possibles:
  - Standalone query operators
  - LINQ Style Query Commands (montré ici)

- Tutoriel:

https://www.queryverse.org/Query.jl/stable/

GadFly

Entrée [38]:
```
using RDatasets, Gadfly

iris = dataset("datasets", "iris")
first(iris, 10)
```

Out[38]: 10 rows × 5 columns

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
	Float64	Float64	Float64	Float64	Categorical...
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa

- Implémente le grammar of graphics utilisé dans ggplot2
- Exemple avec le jeu de données iris
- Tutoriel:

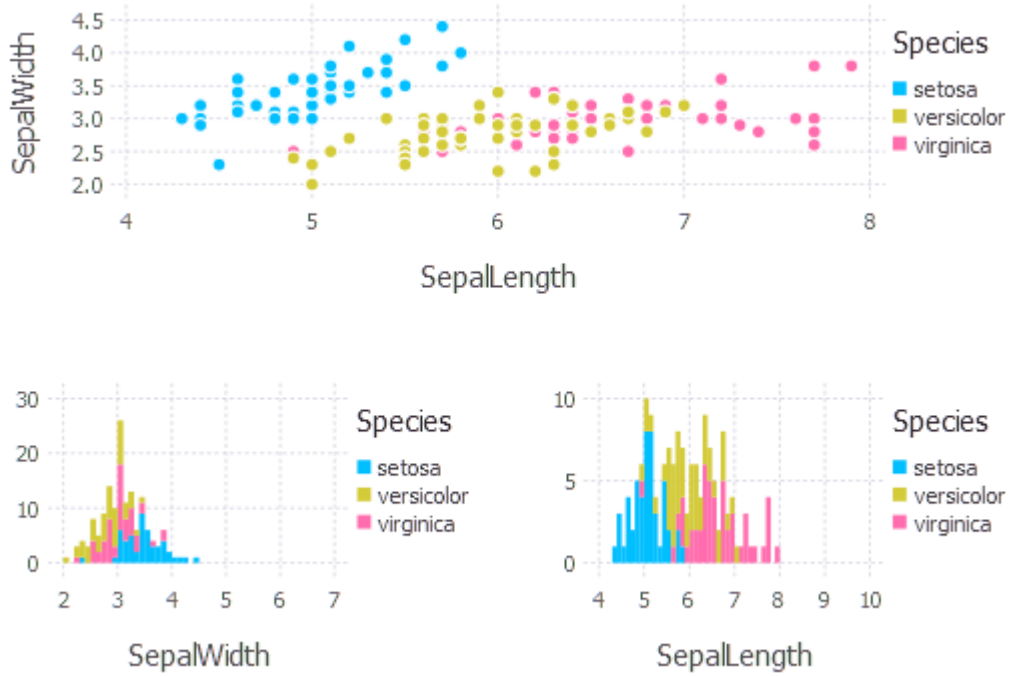
  http://gadflyjl.org/stable/

GadFly: premiers plots

Entrée [45]:

```
p1 = plot(iris,
          x=:SepalLength, y=:SepalWidth,
          color=:Species,
          Geom.point);
p2 = plot(iris,
          x=:SepalWidth, color=:Species,
          Geom.histogram(bincount=50));
p3 = plot(iris,
          x=:SepalLength, color=:Species,
          Geom.histogram(bincount=50));
p_low = hstack(p2, p3);
vstack(p1, p_low)
```

Out[45]:

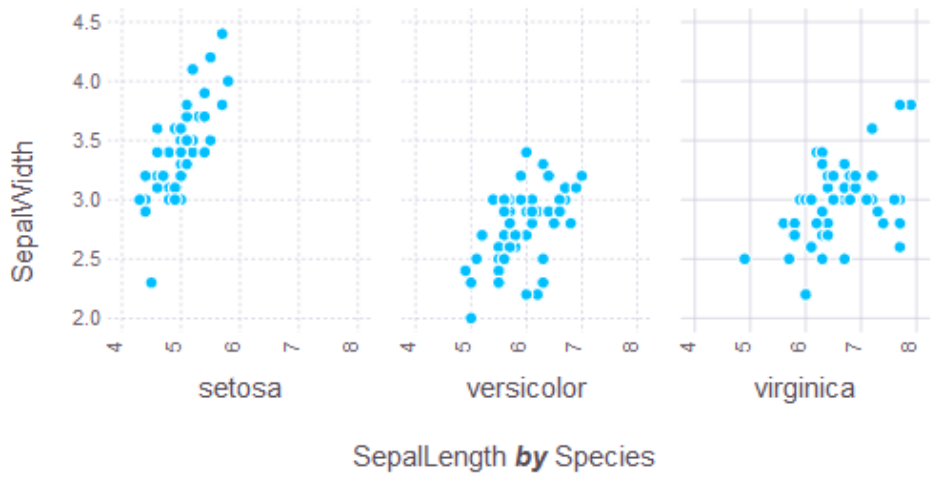


- Il existe une liste de géométrie:

Abline, hline, vline, bar, beeswarm, histogram, point, line, boxplot, density,…

Utilisation d'une grille avec Gadfly

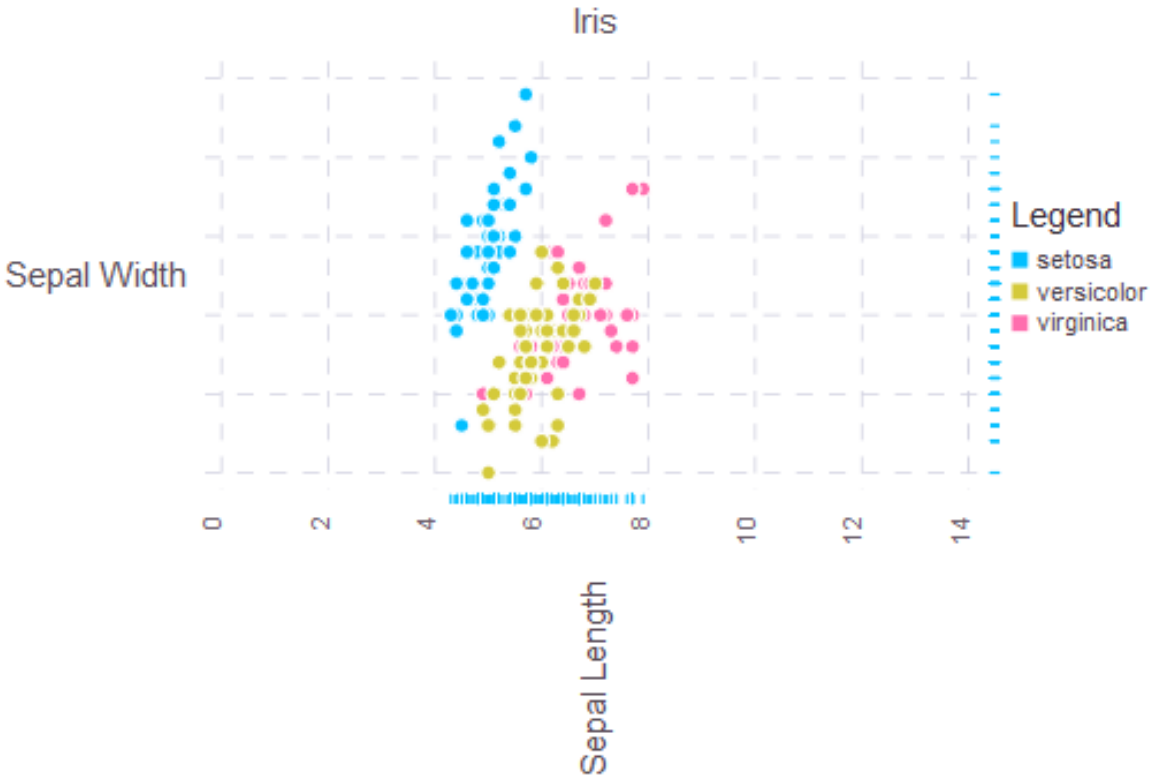
```
using Gadfly, RDatasets
iris = dataset("datasets", "iris")
plot(iris, xgroup="Species", x="SepalLength", y="SepalWidth",
     Geom.subplot_grid(Geom.point))
```



Personalisationd des axes avec Gadfly

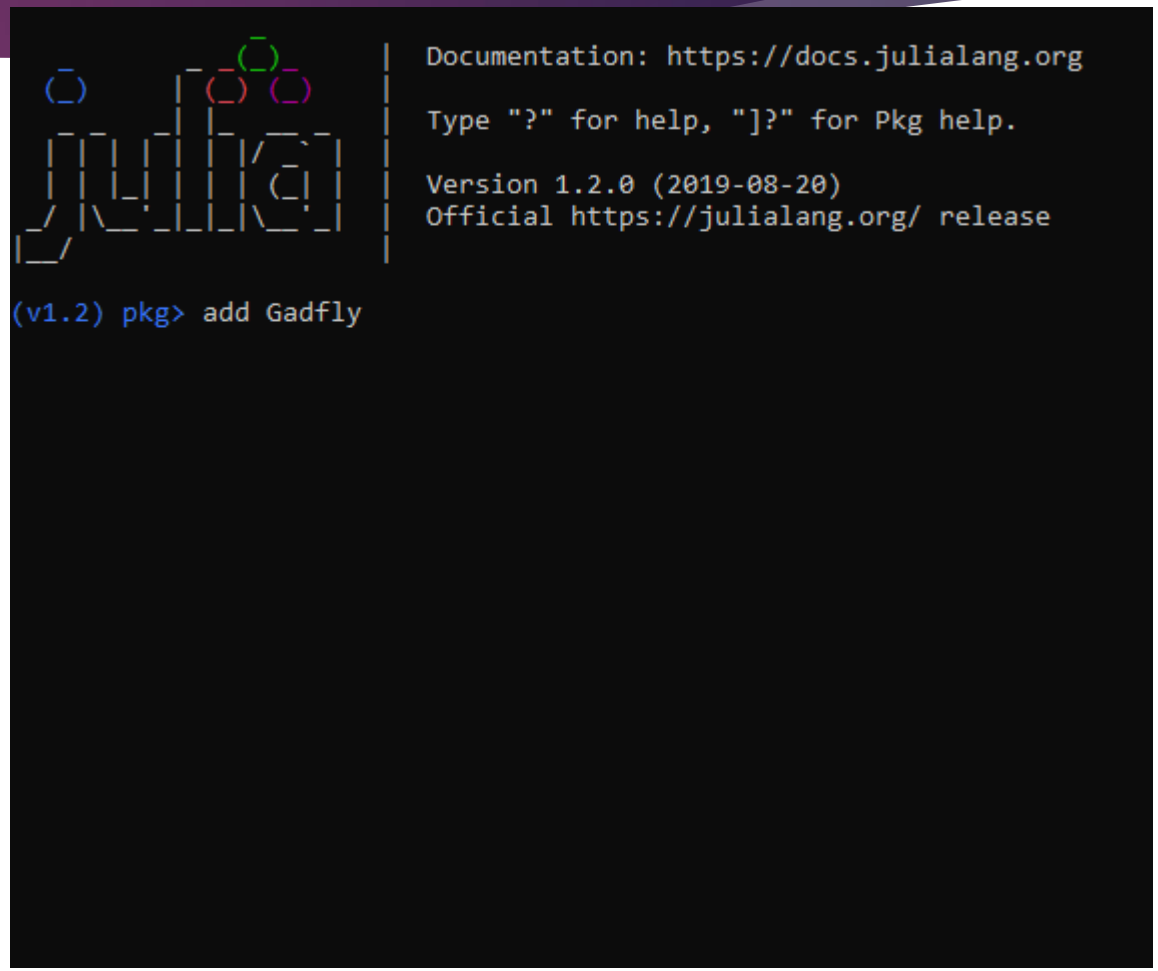
Entrée [51]:

```
plot(iris, x=:SepalLength, y=:SepalWidth, color=:Species,
     Guide.title("Iris"),
     Guide.colorkey(title="Legend"),
     Guide.xrug, Guide.yrug,
     Guide.xlabel("Sepal Length", orientation=:vertical),
     Guide.ylabel("Sepal Width", orientation=:horizontal),
     Guide.yticks(label=false),
     Guide.xticks(label=true, ticks=[2i for i in 0:7], orientation=:vertical)
     )
```



Les packages

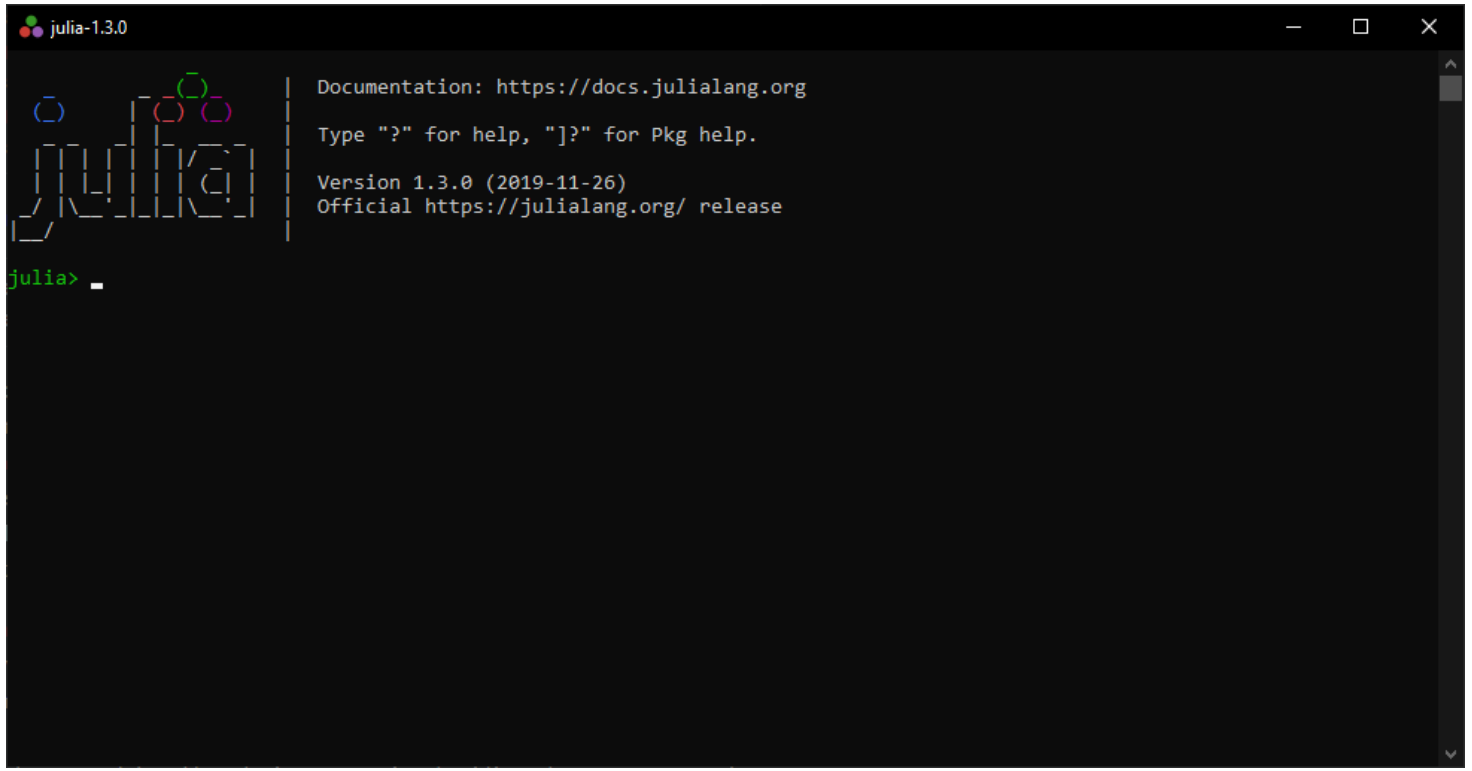
- Pour gérer ces packages il suffit de taper] dans un terminal
- Plusieurs commandes:
  - activate
  - add
  - rm
  - update
  - status
  - ?
- Les packages sont stockés dans homedir() / .julia



Comment utiliser Julia ?

- Utilisation depuis un terminal
- En le téléchargeant depuis:

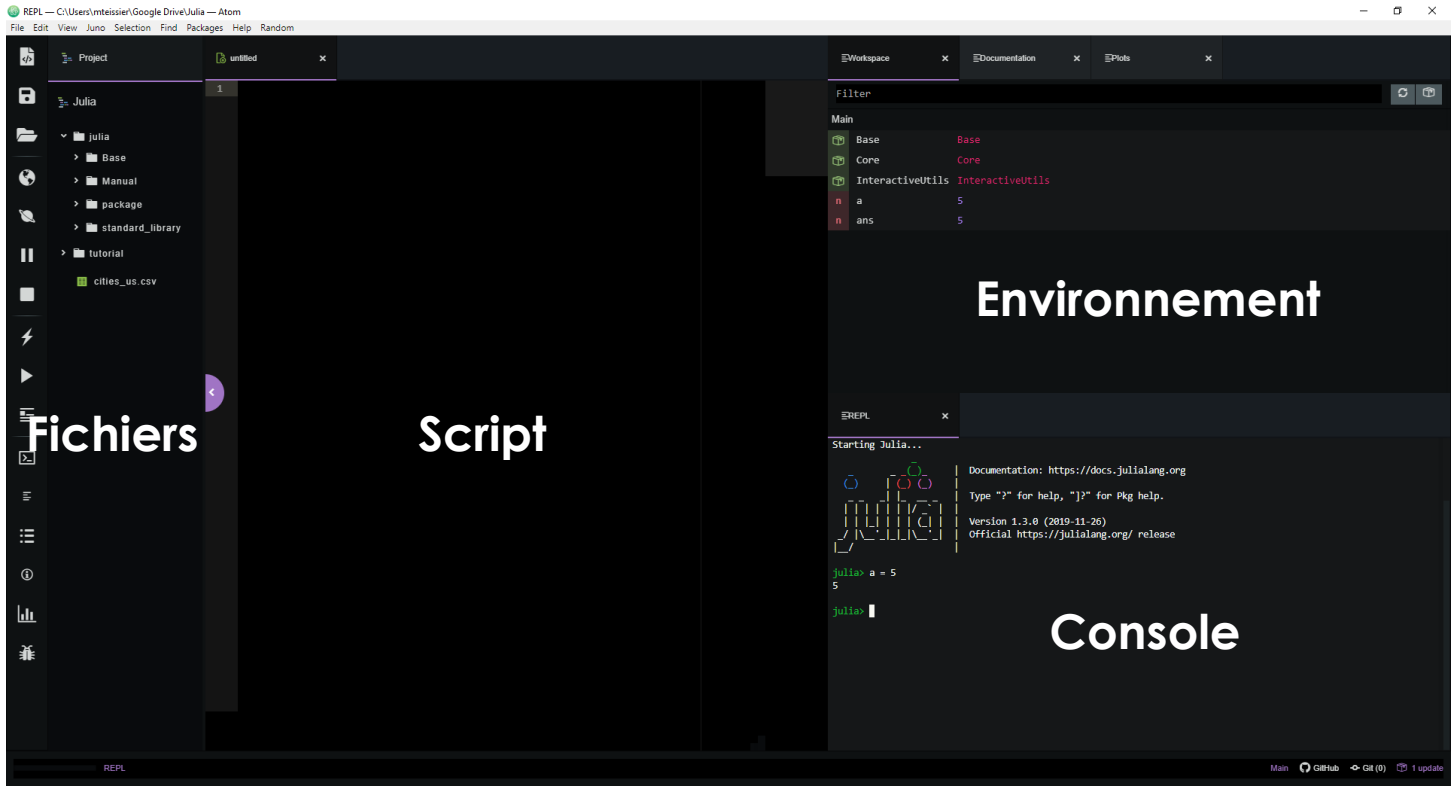
https://julialang.org/downloads/



Comment utiliser Julia ?

- Utilisation d'un IDE (Windows):
  1. Télécharger Julia
  2. Télécharger Atom (https://atom.io/)
  3. Installez le package Juno

- Installation possible sur OS X ou Linux



Merci de votre attention !