OpenTechSchool Beginner Workshop Julia

March 2, 2019 @Co-Up Berlin

Code of Conduct

`http://www.opentechschool.org/code-of-conduct/`

WiFi

`login: co_up  pw: clubmate`

Juliabox (please login/sign-up)

`https://www.juliabox.com/`

Agenda

10 am: Intro

0) Code of Conduct
1) How Julia differs from Python and C
2) Which applications are particularly promising
3) How to get started programming in Julia

10.45 am: We split into **balanced groups** and **work through the tutorials** on `juliabox.org`; coaches help whenever you get stuck.

2 pm: We will be a bit **more creative** and work on several projects, e.g.

- writing our own opentechschool tutorial(s)
- implementing your current project into Julia
- working on more advanced stuff
  `https://etherpad.net/p/Julia_teaching-material`

THE RULES

1) OpenTechSchool Code of Conduct (implicit agreement!)
   - Inclusiveness
   - Friendly, safe and welcoming environment
   - Open [Source/Culture/Tech] Citizenship

THE RULES

1) OpenTechSchool Code of Conduct (implicit agreement!)
   - Inclusiveness
   - Friendly, safe and welcoming environment
   - Open [Source/Culture/Tech] Citizenship
2) And beyond this…

Do

☺ Keep it simple

☺ How can you foster diversity?

☺ Enforce good behavior

Don't

☹ Sales pitch, recruitment

☹ Typing on somebody else's keyboard

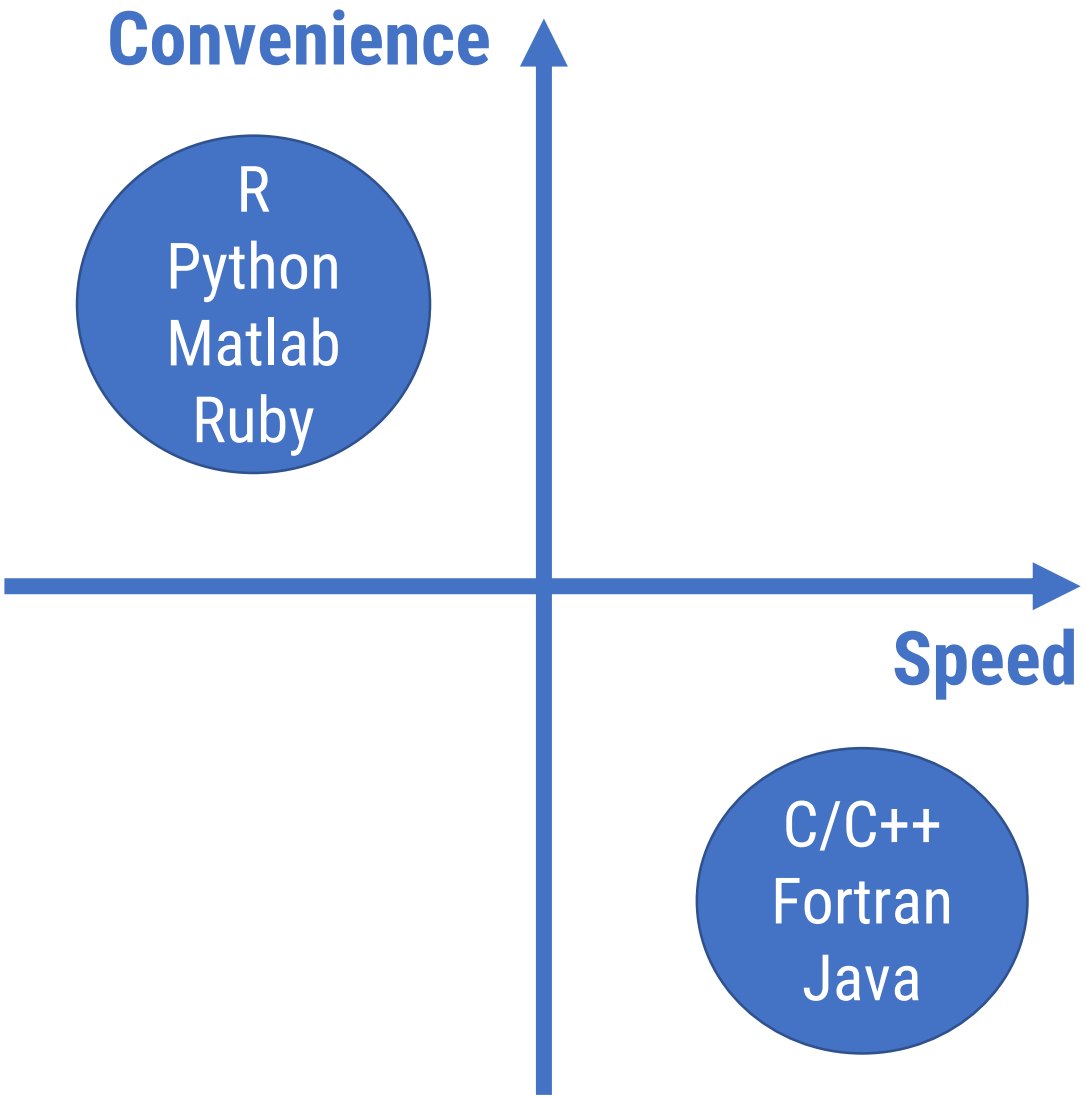
☹ Language bashing

Who are you?

Quickly introduce yourself

C/C++, Python,… Julia?

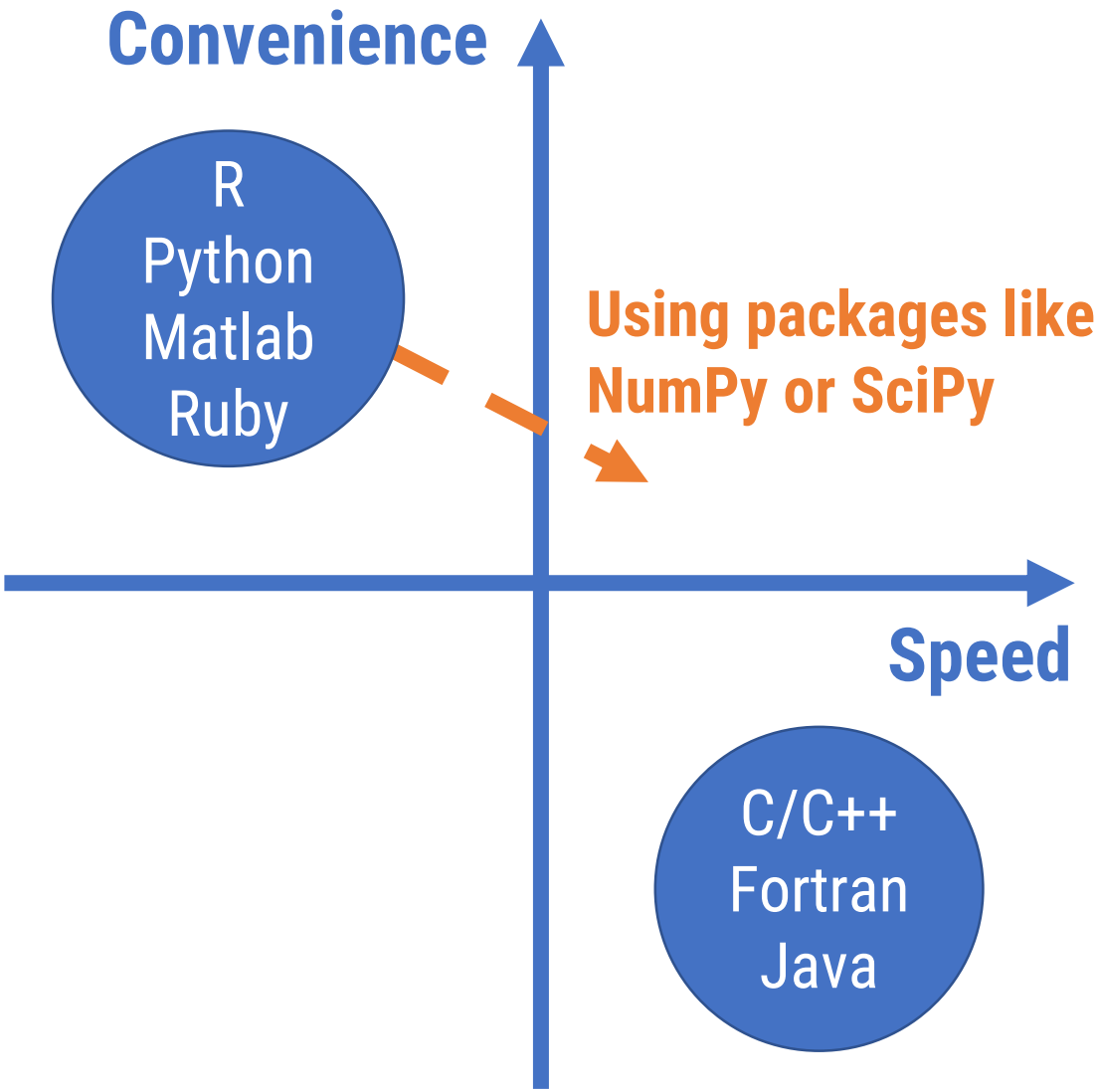
The two-language problem



Typical workflow

1. Develop algorithms and in a language like MATLAB or Python.
2. Rewrite parts of it or the whole thing in a compiled language like C/C++ or Fortran.

The two-language problem



Typical workflow

1. Develop algorithms and in a language like MATLAB or Python.
2. Rewrite parts of it or the whole thing in a compiled language like C++ or Fortran.

Speed vs. convenience – Why?

From high level… …to… …low level language

Source code
= what you type

```
int main() {
int a = 2;
int b = 3;
int c = a + b;
return c;
}
```

Compiler
or
Interpreter

Assembler code
= what it becomes
after optimizing

```
push rbp
mov rbp, rsp
mov DWORD PTR [rbp-4], 2
mov DWORD PTR [rbp-8], 3
mov eax, DWORD PTR [rbp-8]
mov edx, DWORD PTR [rbp-4]
add eax, edx
mov DWORD PTR [rbp-12], eax
mov eax, DWORD PTR [rbp-12]
pop rbp
ret
```

Machine code
= what the
CPU can run

```
55
48 89 E5
C7 45 FC 02
C7 45 F8 03
8B 45 F8
8B 55 FC
01 D0
89 45 F4
8B 45 F4
5D
C3
```

Speed vs. convenience – Why?

From high level… …to… …low level language

Source code
= what you type

```
int main() {

int a = 2;
int b = 3;

int c = a + b;

return c;

}
```

Compiler
or
Interpreter

Assembler code
= what it becomes
after optimizing

```
push rbp
mov rbp, rsp

mov DWORD PTR [rbp-4], 2
mov DWORD PTR [rbp-8], 3
mov eax, DWORD PTR [rbp-8]
mov edx, DWORD PTR [rbp-4]
add eax, edx
mov DWORD PTR [rbp-12], eax

mov eax, DWORD PTR [rbp-12]

pop rbp
ret
```

Machine code
= what the
CPU can run

```
55
48
C
C
8
8
0
89
8B
5D
C3
```

Running a program = simple operations on numbers stored in memory

Speed vs. convenience – Why?

From high level… …to… …low level language

Source code
= what you type

Assembler code
= what it becomes
after optimization

Machine code
= what the
CPU can run

```
push rbp
```

```
return c;
```

```
mov eax, DWO[12]
```

```
pop rbp
ret
```

```
}
```

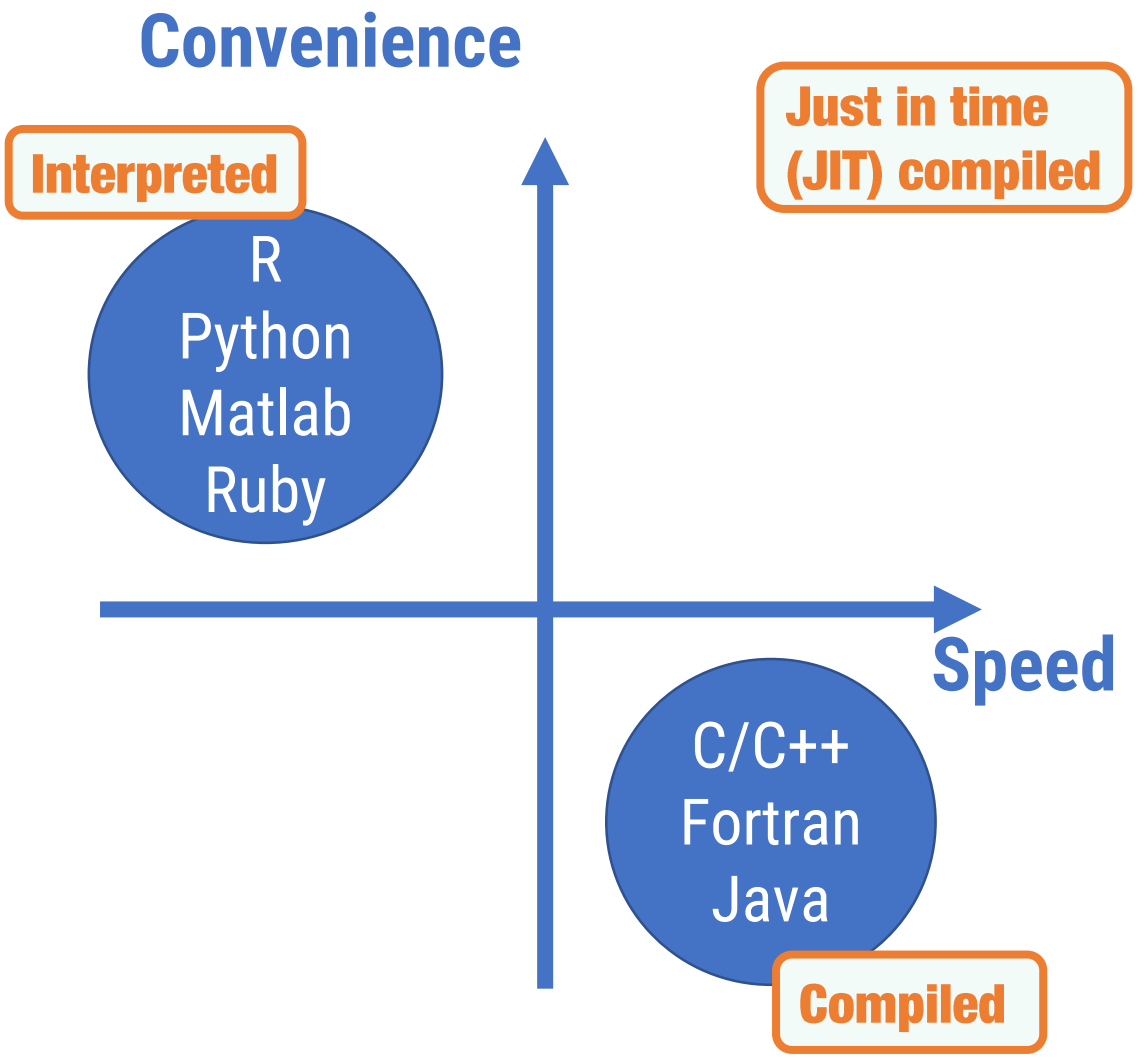
5D
C3

Compiler produces optimized assembler code
Interpreter "just" gets things running

- ⇨ Without knowing the type of a variable, more memory must be reserved (save a float, int…?)
- ⇨ Functions cannot be optimized when input type is unknown (add a matrix, string, integer?)

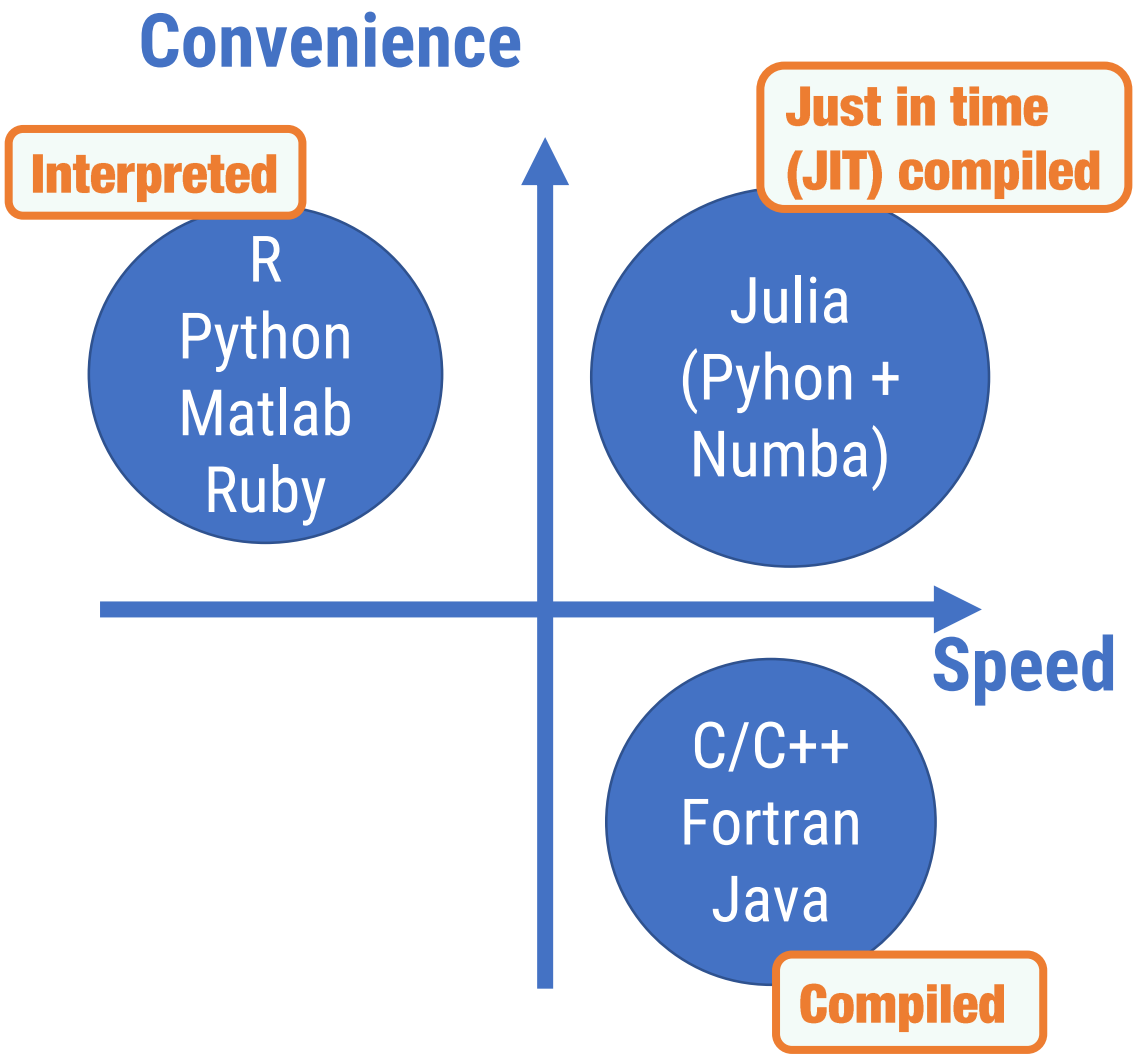
Running a program = simple operations on numbers stored in memory

How to get the best of both worlds in ONE language?



- Some compilation gives a lot of speed

How to get the best of both worlds in ONE language?



- Some compilation gives a lot of speed
- Julia is designed to make this efficient
- Focus on type-stable code

Multiple dispatch

A simple function to square the input

```
f(x) = x * x
```

Definition of the function with the name **f**

Calling the function **f** with the the argument **2**

```
f(2)
```

⇨ **x = 2**

⇨ **Returns the value x * x = 2 * 2 = …**

What do you expect?

A simple function to square the input

```
f(x) = x * x
```

Definition of the function with the name **f**

Calling the function **f** with the the argument **2**

```
f(2)
```

⇨ **x = 2**

⇨ **Returns the value x * x = 2 * 2 = 4**

Calling the function **f** with the the argument **"hello"**

```
f("hello")
```

⇨ **x = "hello"**

⇨ **Returns the value x * x = "hello" * "hello" = …**

What do you expect?

A simple function to square the input

```
f(x) = x * x
```

Definition of the function with the name **f**

Calling the function **f** with the the argument **2**

```
f(2)
```

⇨ **x = 2**

⇨ **Returns the value x * x = 2 * 2 = 4**

Calling the function **f** with the the argument **"hello"**

```
f("hello")
```

⇨ **x = "hello"**

⇨ **Returns the value x * x = "hello" * "hello" = "hellohello"**

Let’s use a different method for Strings

```
f(x) = x * x

f(x::String) = x *
"2"
```

Definition of the function with the name **f**

Definition of what the function **f** does with Strings

Calling the function **f** with the the argument **2**

```
f(2)
```

⇨ **x = 2**

⇨ **Returns the value x * x = 2 * 2 = …**

What do you expect?

Let's use a different method for Strings

```
f(x) = x * x

f(x::String) = x *
"²"
```

Definition of the function with the name **f**

Definition of what the function **f** does with Strings

Calling the function **f** with the the argument **2**

```
f(2)
```

⇨ **x = 2**

⇨ **Returns the value x * x = 2 * 2 = 4**

Calling the function **f** with the the argument **"hello"**

```
f("hello")
```

⇨ **x = "hello"**

⇨ **Returns the value x * "²" = "hello" * "²" = …**

What do you expect?

Multiple dispatch

- A function can have
  different methods for different data types
  ⇨ Actually, think of a function as a collection of methods

Multiple dispatch

- A function can have different methods for different data types
  - ⇨ Actually, think of a function as a collection of methods

- If no limitation is specified, it's the method for all data types (except those that have their special method)
  - ⇨ Do not limit the applicability of your code by unnecessarily specifying the input type
  - ⇨ If you specify the input type, do it generous, e.g., use **`AbstractFloat`** instead of **`Float64`** as this allows to use your code with different precision (AD)

Where does Julia shine?

Performance **and** time of writing the code matters

Julia

Currently NOT the case for

- Supercomputers so expensive that having programmers writhing Fortran code is efficient
- Problems where time doesn’t matter and great solutions exist

New (scientific) algorithms

Julia

Pro

- Writing for-loops is fine
- Composes well with other Julia packages e.g. NeuralNetDiffEq.jl rests on DifferentialEquations.jl, which rests on AutoDiff.jl
- Open source
- Use other languages (Python, R, C)

Con

- Adaptation is slow (e.g. SPSS)
- Existing libraries
- No company like Mathworks behind it

If multiple languages are deadly

Julia

Why?

Your Python code
(e.g. a for-loop)

No optimization beyond this point

Used C libraries
(e.g. Tensorflow)

Your Julia code
(e.g. a for loop)

Jointly optimized

Used Julia
libraries
(e.g. Flux)

Why Not?

Tensorflow focuses on Swift (unless you have a Windows PC)

If code reusability matters ▶ Julia

Why?

Multiple dispatch allows to write generic functions

⇨ Convenient adaptation of algorithms for different hardware

`Array`

`DistributedArrays` (many CPUs)

`CuArray` (with GPUs)

`XRTArray` (with TPUs)

Why Not?

Tensorflow focuses on Swift (unless you have a Windows PC)

Inclusive language from the beginning to the end

Inclusive language from the beginning to the end

Why?

Julia

- is as convenient as Python
- has better access to “deep” concepts than C/C++

Why Not?

Current Julia community = early adopters

⇨ Mainly expert level

⇨ Mainly from fields with “traditionally exclusive mindset”

Getting started

https://youtu.be/y4MMK7JxvZI?t=63

How to get started?

⇨ `juliabox.org`