Introduction to Computational Science in Julia

Adeleke O. Bankole

University of Cambridge
Institute of Computing for Climate Science (ICCS)

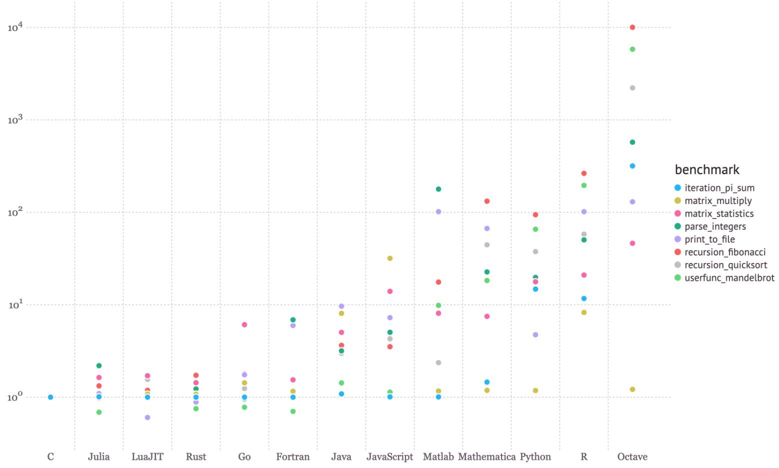
ICCS Summer School
Cambridge, Wednesday 9th July 2025

- Established in 2009, then first appeared in 2012
- Strong **math** support and **numerical** focus
- Package manager integrated into the language
- Fast, dynamic, and high-level syntax
- Reproducible and composable
- General and open source
- Rising popularity and presence in the sciences, particular **climate science** (CliMA), **computational fluid dynamics** (Trixi).

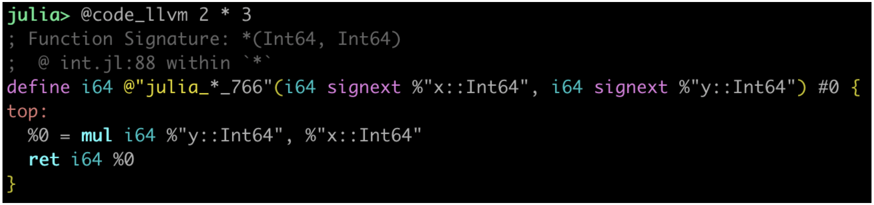
Our aim: give you a crash course introduction

GitHub link to material:

- **Performance**: High performance via just-in-time (JIT) compilation to LLVM



What happens in this simple code expression, $2 + 3$?



```
julia> @code_llvm 2 * 3
; Function Signature: *(Int64, Int64)
;  @ int.jl:88 within `*`
define i64 @"julia_*_766"(i64 signext %"x::Int64", i64 signext %"y::Int64") #0 {
top:
  %0 = mul i64 %"y::Int64", %"x::Int64"
  ret i64 %0
}
```

JIT

- Specializes on types of function arguments
- When a function is called, it compiles efficient machine code
- Existing machine code is reused, if same function is called again

Multi-paradigm language

- **Imperative**: Sequence of statements and definitions, providing a step-by-step sequence of commands

$$y = 5$$
$$y = y + 2$$

- **Functional**: Functions can be treated as data, i.e, returned or can be passed as arguments

```julia
function mkConstantFunction(x)
  function inner(_y)
    x
  end
end
```

```julia
function applyTwice(f, x)
   f(f(x))
end
```

Dynamically typed

```
function plusInt(x :: Int)
  x + 1
end
```

```
function plusGen(x)
  x + 1
end
```

- Type checking is performed at runtime, rather than at compile time
- We can explicitly write signature types

Type stability: A code is type-stable if all input and output variables have a concrete type, either by explicit declaration or by inference from the compiler.

Which function gets executed when a generic function is called for a given set of input arguments?

Multiple dispatch

- Allows execution of different versions of function for different types
- Depending on the types of the arguments
- Chooses the most specialized option
- This provides more flexibility and efficiency

```julia
function scale(x :: Int, y :: Int) :: Int
  x * y
end

function scale(x :: Int, y :: String) :: String
  join([y for i in 1:x])
end
```

Learning Outcomes

- Understand the core ideas of Julia programming language
- Use Julia to solve simple numerical programming tasks
- Working with Julia's type systems and data structures
- Use **multiple dispatch**, including for overloading programs
- Put these ideas together to use in your models

Style: code along plus exercises

Pluto.jl

- Similar to Jupyter notebook
- **Interactive and reactive**: if you change a cell, all the cells depending on it will get run again.
- **Reproducible**: someone else can run your notebook
- Evaluate cell via Shift + Enter

Pluto.jl

Setup, assuming Julia is installed

- From Julia prompt
- Enter package mode, press:]
- Type: add Pluto
- Exit package mode, press: backspace or Ctrl C
- Now back in Julia prompt, type: using Pluto, or import Pluto
- Start Pluto, type: Pluto.run()