给 Julia 开发者的入门教程

看教程之前

以下内容不会在教程中涉及，但是非常重要。

1. 如何配置 Julia 语言环境 (Julia, VSCode, Revise, 软件源)
    - How to develop a Julia package - Chris Rackauchas
    - Julia 模块-远程开发 - 王至宏
2. 您最好对 Git 和 GitHub 有基本的了解， 以便理解 Julia 的软件包管理系统。 相关资料：Missing Semester

离线使用此教程的小贴士：你需要配置 Pluto notebook 以在便本地打开该教程， 该教程将会上传到会议网站。

会议网址是： https://cn.julialang.org/meetup-website/2022/

会议网站

调查

你用什么编程语言?

Julia 是什么样的语言？

源代码开放的现代高性能语言

Julia 于2012年在 MIT 诞生。Julia 语言的解释/编译器的源代码是开放的，被托管在 Github 仓库 中，其软件协议为可商用的 MIT 协议。 不仅 Julia 语言如此， 大多 Julia 的软件包系统也依托 Github 管理， 其协议也大多为开源。

Julia 语言被设计出来的目的是为了兼顾代码执行速度与开发效率.

- 执行速度: C, C++, Fortran
- 开发效率: Python, Matlab

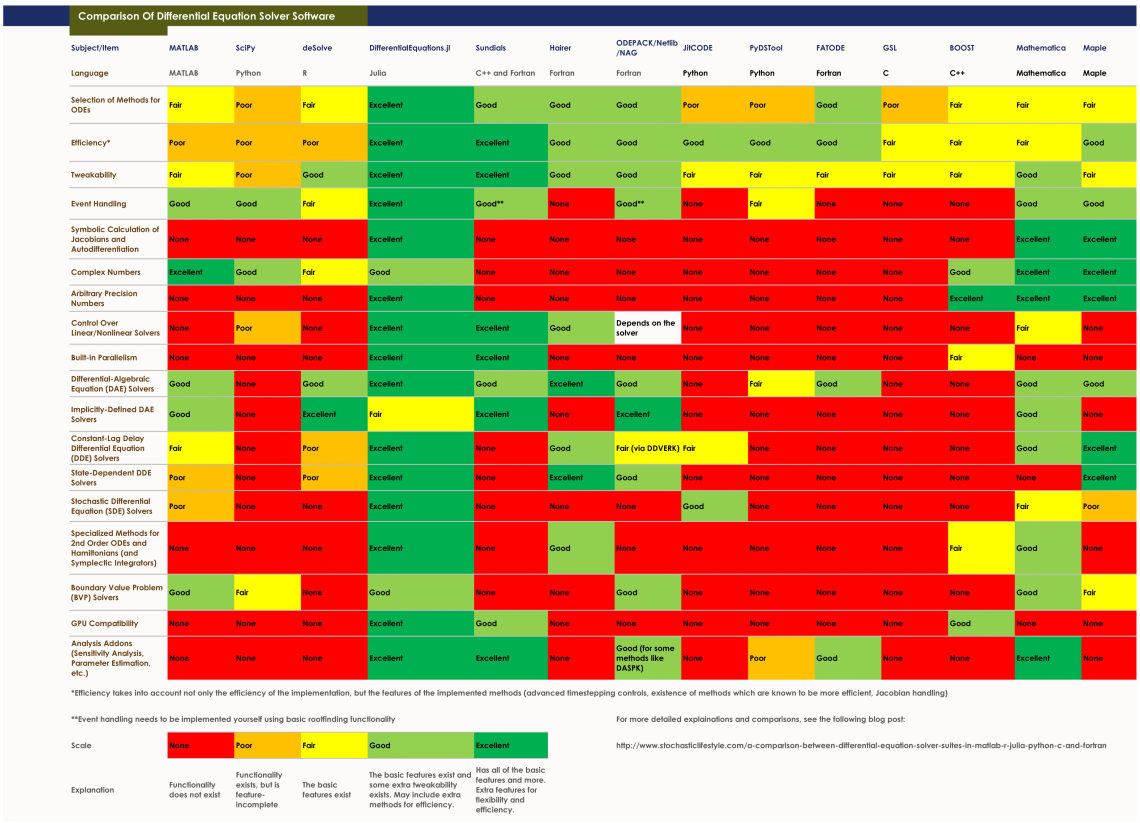
Julia 的软件生态

所有的软件包都可以在JuliaHub上找到相关统计。

科学计算生态

SciML ecosystem

微分方程的求解



Comparison Of Differential Equation Solver Software

Subject/Item	MATLAB	SciPy	deSolve	DifferentialEquations.jl	Sundials	Hairer	ODEPACK/Netlib /NAG	JITCODE	PyDSTool	FATODE	GSL	BOOST	Mathematica	Maple
Language	MATLAB	Python	R	Julia	C++ and Fortran	Fortran	Fortran	Python	Python	Fortran	C	C++	Mathematica	Maple
Selection of Methods for ODEs	Fair	Poor	Fair	Excellent	Good	Good	Good	Poor	Poor	Good	Poor	Fair	Fair	Fair
Efficiency*	Poor	Poor	Poor	Excellent	Excellent	Good	Good	Good	Good	Good	Fair	Fair	Fair	Good
Tweakability	Fair	Poor	Good	Excellent	Excellent	Good	Good	Fair	Fair	Fair	Fair	Fair	Good	Fair
Event Handling	Good	Good	Fair	Excellent	Good**	None	Good**	None	Fair	None	None	None	Good	Good
Symbolic Calculation of Jacobians and Autodifferentiation	None	None	None	Excellent	None	None	None	None	None	None	None	None	Excellent	Excellent
Complex Numbers	Excellent	Good	Fair	Good	None	None	None	None	None	None	None	Good	Excellent	Excellent
Arbitrary Precision Numbers	None	None	None	Excellent	None	None	None	None	None	None	None	Excellent	Excellent	Excellent
Control Over Linear/Nonlinear Solvers	None	Poor	None	Excellent	Excellent	Good	Depends on the solver	None	None	None	None	None	Fair	None
Built-in Parallelism	None	None	None	Excellent	Excellent	None	None	None	None	None	None	Fair	None	None
Differential-Algebraic Equation (DAE) Solvers	Good	None	Good	Excellent	Good	Excellent	Good	None	Fair	Good	None	None	Good	Good
Implicitly-Defined DAE Solvers	Good	None	Excellent	Fair	Excellent	None	Excellent	None	None	None	None	None	Good	None
Constant-Lag Delay Differential Equation (DDE) Solvers	Fair	None	Poor	Excellent	None	Good	Fair (via DDVERK)	Fair	None	None	None	None	Good	Excellent
State-Dependent DDE Solvers	Poor	None	Poor	Excellent	None	Excellent	Good	None	None	None	None	None	None	Excellent
Stochastic Differential Equation (SDE) Solvers	Poor	None	None	Excellent	None	None	None	Good	None	None	None	None	Fair	Poor
Specialized Methods for 2nd Order ODEs and Hamiltonians (and Symplectic Integrators)	None	None	None	Excellent	None	Good	None	None	None	None	None	Fair	Good	None
Boundary Value Problem (BVP) Solvers	Good	Fair	None	Good	None	None	Good	None	None	None	None	None	Good	Fair
GPU Compatibility	None	None	None	Excellent	Good	None	None	None	None	None	None	Good	None	None
Analysis Addons (Sensitivity Analysis, Parameter Estimation, etc.)	None	None	None	Excellent	Excellent	None	Good (for some methods like DASPK)	None	Poor	Good	None	None	Excellent	None

*Efficiency takes into account not only the efficiency of the implementation, but the features of the implemented methods (advanced timestepping controls, existence of methods which are known to be more efficient, Jacobian handling)

**Event handling needs to be implemented yourself using basic rootfinding functionality

For more detailed explainations and comparisons, see the following blog post:

http://www.stochasticlifestyle.com/a-comparison-between-differential-equation-solver-suites-in-matlab-r-julia-python-c-and-fortran

Scale	None	Poor	Fair	Good	Excellent
Explanation	Functionality does not exist	Functionality exists, but is feature-incomplete	The basic features exist	The basic features exist and some extra tweakability exists. May include extra methods for efficiency.	Has all of the basic features and more. Extra features for flexibility and efficiency.

JuMP ecosystem

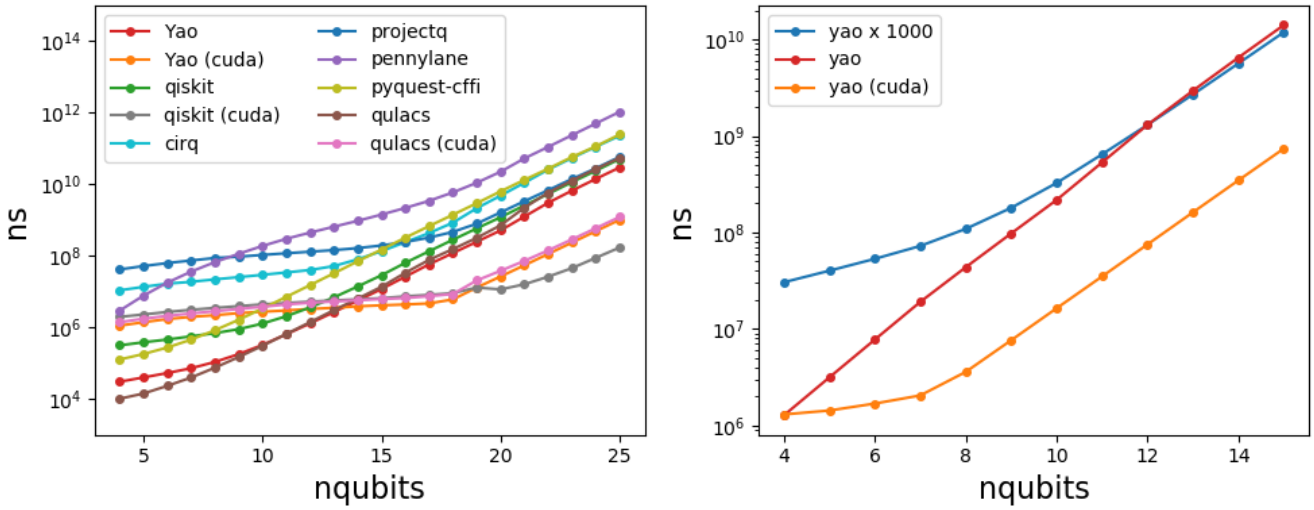
Linear Programming, Mixed Integer Programming, Quadratic Programming 等。

Table I *Time (sec.) to generate each model and pass it to the solver; a comparison between JuMP and existing commercial and open-source modeling languages. The* `lqcp` *instances have quadratic objectives and linear constraints. The* `fac` *instances have linear objectives and conic-quadratic constraints.*

		Commercial			Open-source		
Instance	**JuMP**	GRB/C++	AMPL	GAMS	Pyomo	CVX	YALMIP
lqcp-500	8	2	2	2	55	6	8
lqcp-1000	11	6	6	13	232	48	25
lqcp-1500	15	14	13	41	530	135	52
lqcp-2000	22	26	24	101	>600	296	100
fac-25	7	0	0	0	14	>600	533
fac-50	9	2	2	3	114	>600	>600
fac-75	13	5	7	11	391	>600	>600
fac-100	24	12	18	29	>600	>600	>600

Yao ecosystem

Quantum Computing

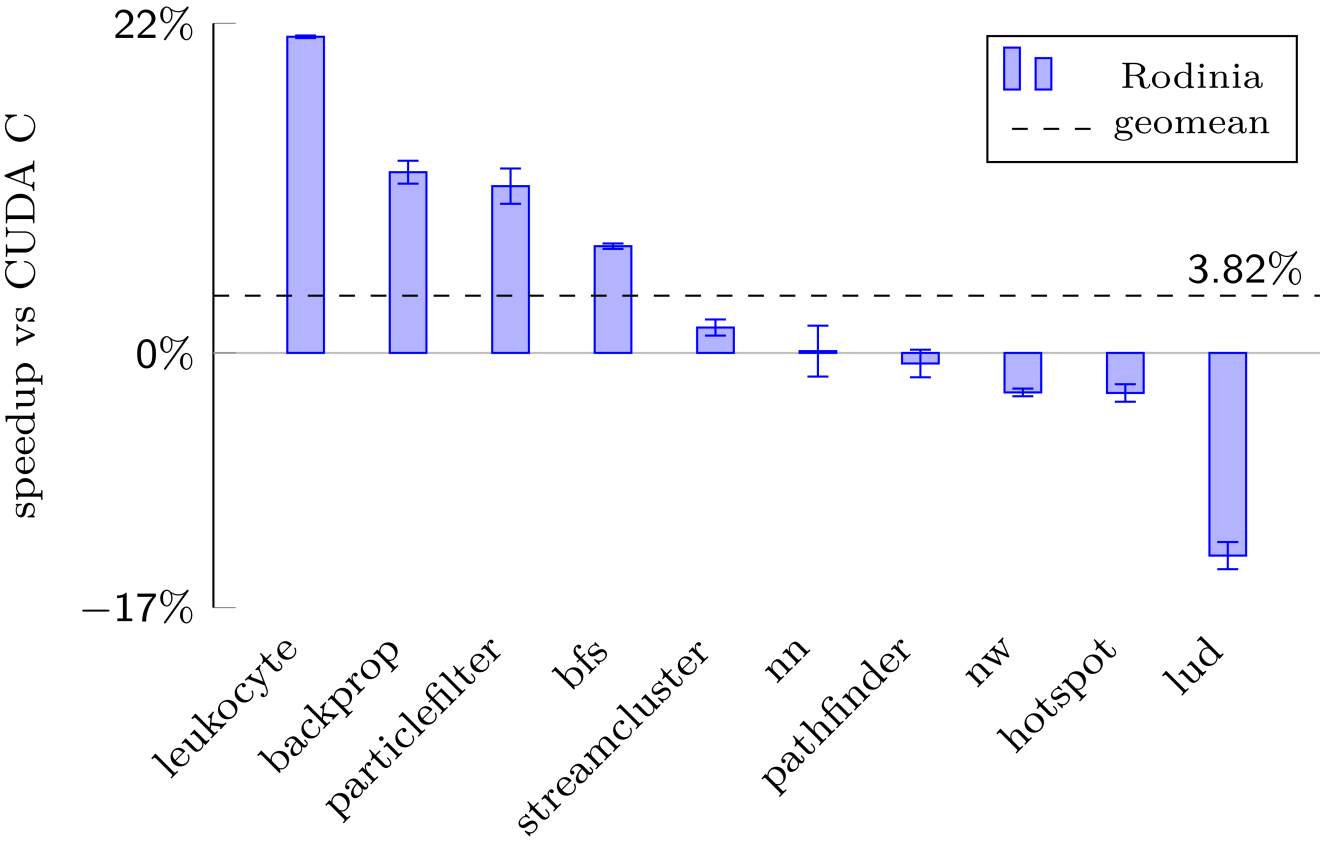


还有更多有趣的软件生态，包括 BioJulia, JuliaDynamics, EcoJulia, JuliaAstro, QuantEcon.

高性能计算生态

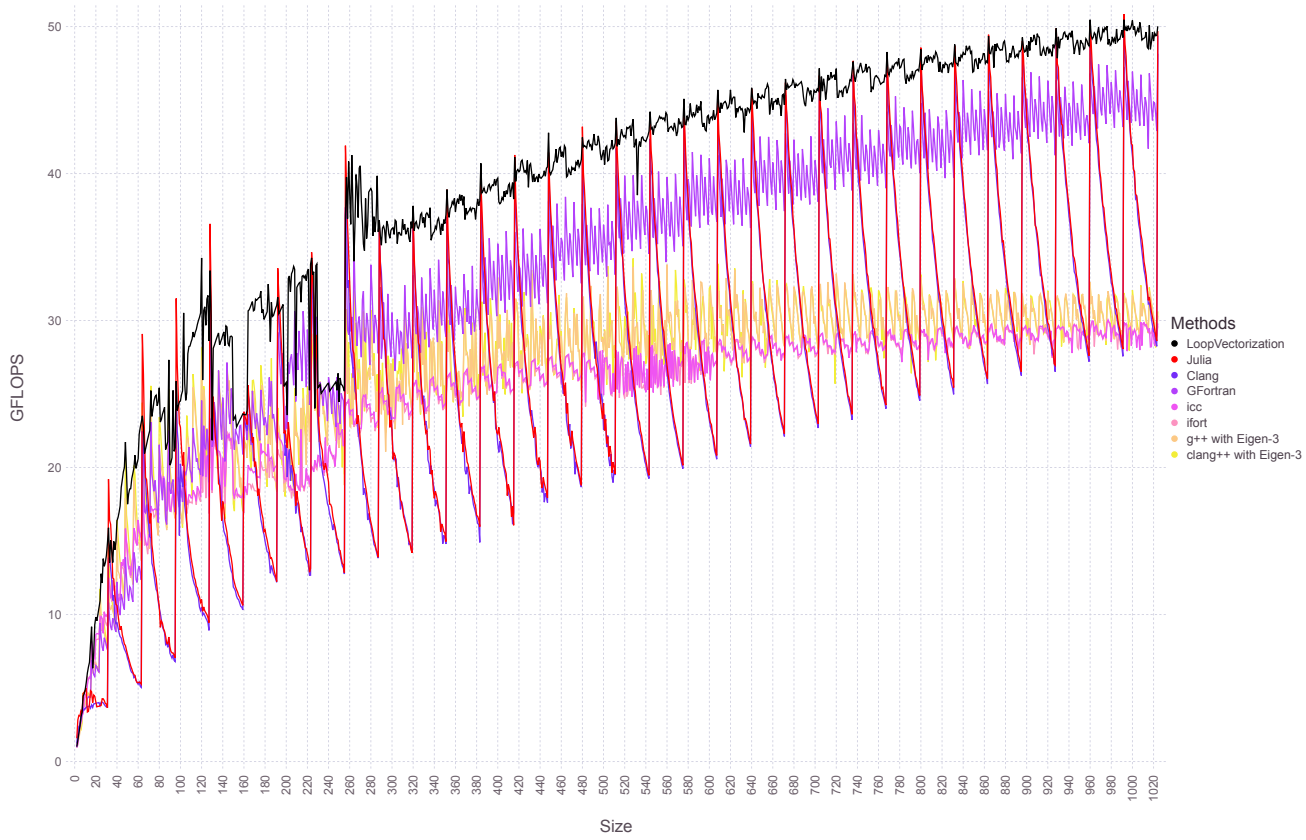
CUDA ecosystem

CUDA programming in Julia.

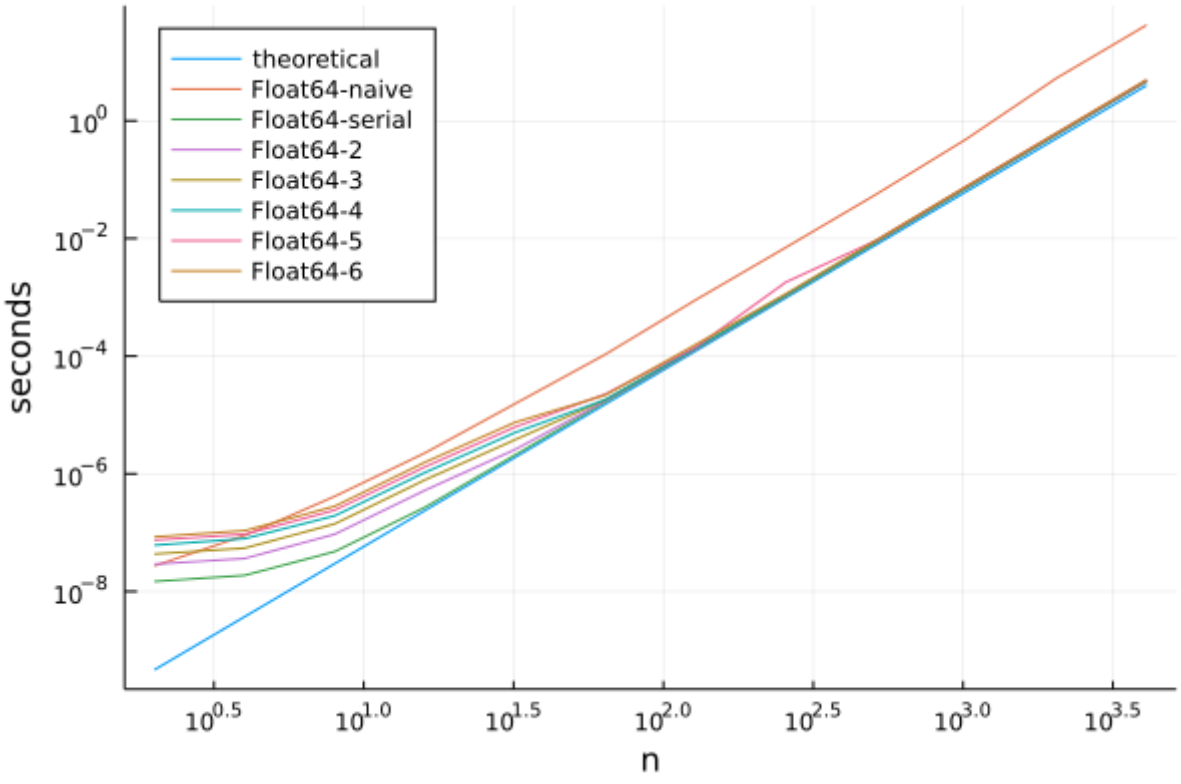


LoopVectorization ecosystem

Macro(s) for vectorizing loops (SIMD).



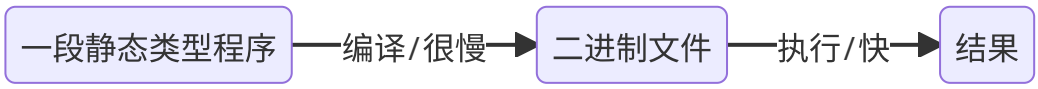
TropicalGEMM: A BLAS for tropical numbers.



Julia 为什么快？

编译语言快的秘诀

静态类型程序的执行很快，因为类型信息被提前知道就可以被高效的编译。




```
FILE: clib/demo.c
--------------------------------------------------------------------------------
#include <stddef.h>
int c_factorial(size_t n) {
    int s = 1;
    for (size_t i=1; i<=n; i++) {
        s *= i;
    }
    return s;
}
```

```
showfile("clib/demo.c")
```

```
Process(`gcc clib/demo.c -fPIC -O3 -msse3 -shared -o clib/demo.so`, ProcessExited(0))
```

```
# compile to a shared library by piping C_code to gcc;
# (only works if you have gcc installed)
run(`gcc clib/demo.c -fPIC -O3 -msse3 -shared -o clib/demo.so`)
```

```
using Libdl
```

```
c_factorial (generic function with 1 method)
```

```
c_factorial(x) = Libdl.@ccall "clib/demo".c_factorial(x::Csize_t)::Int
```

```
3628800
```

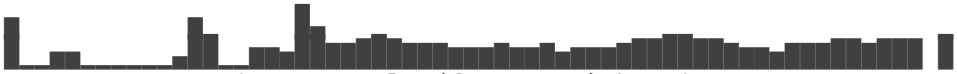
```
c_factorial(10)
```

```
0
```

```
c_factorial(1000)
```

```
using BenchmarkTools
```

```
BenchmarkTools.Trial: 10000 samples with 121 evaluations.
 Range (min … max):  750.182 ns …  1.152 μs  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     812.909 ns              ┊ GC (median):    0.00%
 Time  (mean ± σ):   809.759 ns ± 32.595 ns  ┊ GC (mean ± σ):  0.00% ± 0.00%
```



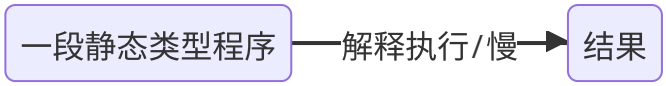

```
Memory estimate: 0 bytes, allocs estimate: 0.
```

```
@benchmark c_factorial(1000)
```

learn more about calling C code in Julia

解释语言方便的秘诀

动态类型的语言不需要被编译




```
using PyCall
```

```
# py"..." is a string literal, it is defined as a special macro: @py_str
py"""
def factorial(n):
    x = 1
    for i in range(1, n+1):
        x = x * i
    return x
"""
```

```
402387260077093773543702433923003985719374864210714632543799910429938512398629020592044200
```

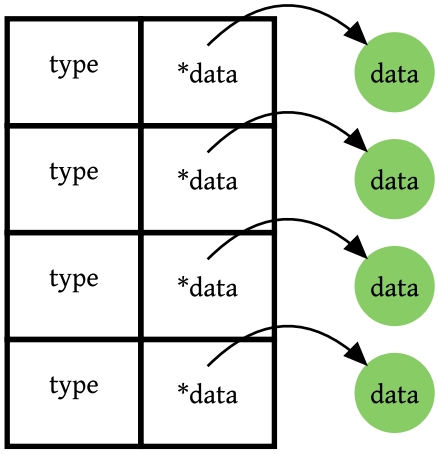
```
py"factorial"(1000)
```

```
9223372036854775807
```

```
# `typemax` 可以获取类型的最大值
typemax(Int)
```

🤔

但由于数据没有固定的类型，解释执行的语言必须用一个 `Box(type, *data)` 来表示一个数据。




```
BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):  294.900 μs … 10.239 ms  ┊ GC (min … max): 0.00% … 44.77%
 Time  (median):     320.793 μs              ┊ GC (median):    0.00%
 Time  (mean ± σ):   319.717 μs ± 99.780 μs  ┊ GC (mean ± σ):  0.14% ±  0.45%
```

```
 295 μs          Histogram: log(frequency) by time        346 μs <

 Memory estimate: 7.45 KiB, allocs estimate: 11.
```

```
• @benchmark $(py"factorial")(1000)
```

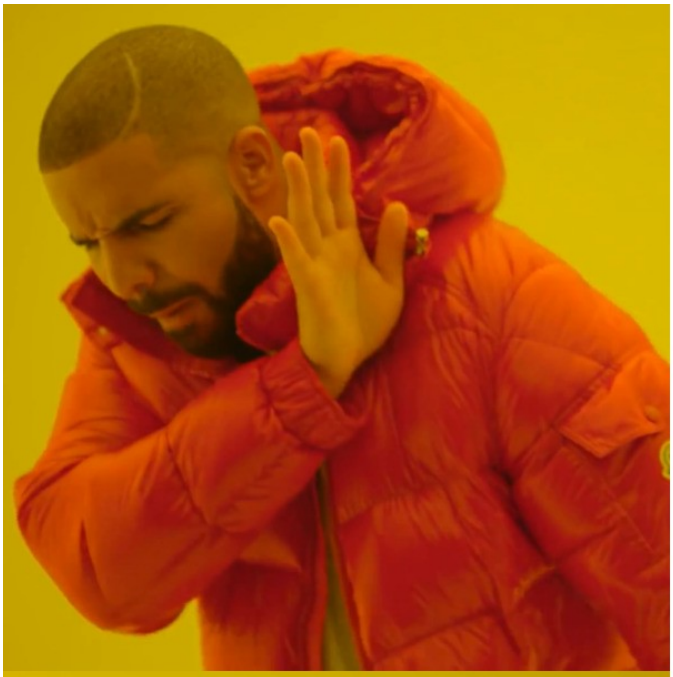
双语言 Python & C++ 的问题?

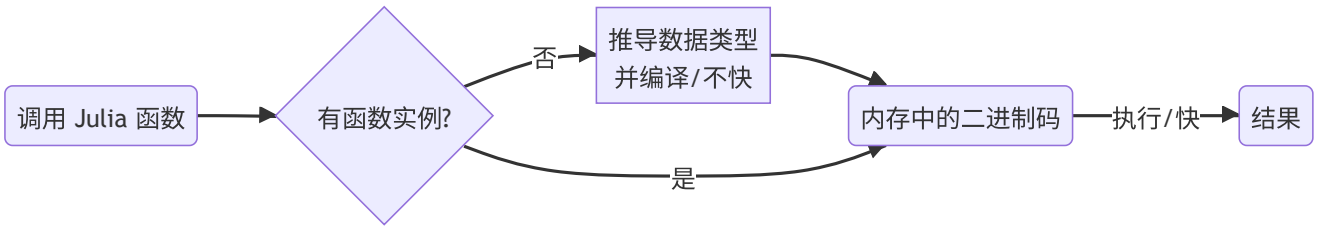
可维护性变差

- 配置 setup 文件更加复杂, 平台移植性变差，
- 培养新人成本过高,

非常适合张量运算, 但很多程序抽象发生在底层

- 蒙特卡洛 (Monte Carlo) 和模拟退火 (Simulated Annealing) 方法, 频繁多变的随机数生成和采样
- 范型张量网络 (Generic Tensor Network), 张量中的标量类型的基本元素非实数, 而是 tropical number 或者有限域 (Finite Field Algebra)
- 组合优化中的分支界定法 (branching)





Julia 函数被编译的过程

1. 拿到一段 Julia 程序

```
jlfactorial (generic function with 1 method)
```

```
function jlfactorial(n::Int)
    x = 1
    for i in 1:n
        x = x * i
    end
    return x
end
```

2. 当遇到调用，在 Julia 的中间表示 (Intermediate Representation) 上推导数据类型

```
MethodInstance for Main.var"workspace#13".jlfactorial(::Int64)
  from jlfactorial(n::Int64) in Main.var"workspace#13" at /home/leo/jcode/notebooks
Arguments
  #self#::Core.Const(Main.var"workspace#13".jlfactorial)
  n::Int64
Locals
  @_3::Union{Nothing, Tuple{Int64, Int64}}
  x::Int64
  i::Int64
Body::Int64
1 ─       (x = 1)
│   %2  = (1:n)::Core.PartialStruct(UnitRange{Int64}, Any[Core.Const(1), Int64])
│         (@_3 = Base.iterate(%2))
│   %4  = (@_3 === nothing)::Bool
│   %5  = Base.not_int(%4)::Bool
└──       goto #4 if not %5
2 ┄ %7  = @_3::Tuple{Int64, Int64}
│         (i = Core.getfield(%7, 1))
```

```
with_terminal() do
    @code_warntype jlfactorial(10)
end
```

3. 将带类型的程序编译到 LLVM 中间表示上

LLVM 是很多语言的后端，如 Julia, Rust, Swift, Kotlin 等.

```
;  @ /home/leo/jcode/notebooks/julia/juliatutorial.jl#==#13bcf3d6-2418-46e1-acde-05
define i64 @julia_jlfactorial_6730(i64 signext %0) #0 {
top:
;  @ /home/leo/jcode/notebooks/julia/juliatutorial.jl#==#13bcf3d6-2418-46e1-acde-05
; ┌ @ range.jl:5 within `Colon`
; │┌ @ range.jl:393 within `UnitRange`
; ││┌ @ range.jl:400 within `unitrange_last`
     %.inv = icmp sgt i64 %0, 0
     %. = select i1 %.inv, i64 %0, i64 0
; └└└
  br i1 %.inv, label %L18.preheader, label %L35

L18.preheader:                                    ; preds = %top
;  @ /home/leo/jcode/notebooks/julia/juliatutorial.jl#==#13bcf3d6-2418-46e1-acde-05
  %min.iters.check = icmp ult i64 %., 16
  br i1 %min.iters.check, label %L18, label %vector.ph

vector.ph:                                        ; preds = %L18.preheader
```

```
with_terminal() do
    @code_llvm jlfactorial(10)
end
```

4. LLVM 在这个中间表示的基础上生成高性能的汇编/二进制码

```
    .text
    .file   "jlfactorial"
    .section    .rodata.cst8,"aM",@progbits,8
    .p2align    3                               # -- Begin function julia_jlfactori
.LCPI0_0:
    .quad   1                               # 0x1
.LCPI0_2:
    .quad   4                               # 0x4
.LCPI0_3:
    .quad   8                               # 0x8
.LCPI0_4:
    .quad   12                              # 0xc
.LCPI0_5:
    .quad   16                              # 0x10
    .section    .rodata.cst32,"aM",@progbits,32
    .p2align    5
.LCPI0_1:
    .quad   1                               # 0x1
```

```
with_terminal() do
    @code_native jlfactorial(10)
end
```

```
0
```

```
jlfactorial(1000)
```

```
BenchmarkTools.Trial: 10000 samples with 355 evaluations.
 Range (min … max):  255.899 ns … 431.341 ns  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     282.200 ns               ┊ GC (median):    0.00%
 Time  (mean ± σ):   286.554 ns ±  20.196 ns  ┊ GC (mean ± σ):  0.00% ± 0.00%
```

```
 256 ns        Histogram: log(frequency) by time        353 ns <

 Memory estimate: 0 bytes, allocs estimate: 0.
```

```
@benchmark jlfactorial(x) setup=(x=1000)
```

案例分析： 查看 Method instances

函数实例 (method instance)： 内存中，一个针对特定输入类型的函数被编译后的二进制码。

有时候类型无法在编译期间被完全定下来。

```
MethodInstance for (::Main.var"workspace#13".var"#unstable#3")(::Int64)
  from (::Main.var"workspace#13".var"#unstable#3")(x) in Main.var"workspace#13" at /h
Arguments
  #self#::Core.Const(Main.var"workspace#13".var"#unstable#3"())
  x::Int64
Body::Union{Float64, Int64}
1 ─ %1 = (x > 3)::Bool
└──      goto #3 if not %1
2 ─      return 1.0
3 ─      return 3
```

```
with_terminal() do
    unstable(x) = x > 3 ? 1.0 : 3
    @code_warntype unstable(4)
end
```

多重派发

Julia 有很多特别之处，在此列举一个其中最重要的一点

Mutliple Dispatch

多重 派发

函数应不应该属于对象？

假设我们想在 python 中实现一个加法。

```
class X:
  def __init__(self, num):
    self.num = num

  def __add__(self, other_obj):
    return X(self.num+other_obj.num)

  def __radd__(self, other_obj):
    return X(other_obj.num + self.num)

  def __str__(self):
    return "X = " + str(self.num)

class Y:
  def __init__(self, num):
    self.num = num

  def __radd__(self, other_obj):
    return Y(self.num+other_obj.num)

  def __str__(self):
    return "Y = " + str(self.num)

print(X(3) + Y(5))


print(Y(3) + X(5))
```

```
# Julian style
struct X{T}
    num::T
end
```

```
struct Y{T}
    num::T
end
```

```
Base.:(+)(a::X, b::Y) = X(a.num + b.num)
```

```
Base.:(+)(a::Y, b::X) = X(a.num + b.num)
```

```
Base.:(+)(a::X, b::X) = X(a.num + b.num)
```

```
Base.:(+)(a::Y, b::Y) = Y(a.num + b.num)
```

```
X(8)
```

```
X(3) + Y(5)
```

```
X(8)
```

```
Y(3) + X(5)
```

现在我把这些代码打包做成了一个package，由个人问我他其实有个C，想拓展这个加法。

```
class Z:
  def __init__(self, num):
    self.num = num

  def __add__(self, other_obj):
    return Z(self.num+other_obj.num)

  def __radd__(self, other_obj):
    return Z(other_obj.num + self.num)

  def __str__(self):
    return "Z = " + str(self.num)

print(X(3) + Z(5))

print(Z(3) + X(5))
```

```
struct Z{T}
    num::T
end
```

```
Base.:(+)(a::X, b::Z) = Z(a.num + b.num)
```

```
Base.:(+)(a::Z, b::X) = Z(a.num + b.num)
```

```
Base.:(+)(a::Y, b::Z) = Z(a.num + b.num)
```

```
Base.:(+)(a::Z, b::Y) = Z(a.num + b.num)
```

```
Base.:(+)(a::Z, b::Z) = Z(a.num + b.num)
```

```
Z(8)
```

```
X(3) + Z(5)
```

```
Z(8)
```

```
Z(3) + Y(5)
```

Julia 的函数空间有指数大！

假如 f 有 k 个参数，类型空间一共定义了t个类型，请问函数空间有多大？

```
f(x::T1, y::T2, z::T3...)
```

如果是 Python 呢？

```
class T1:
    def f(self, y, z, ...):
        self.num = num
```

Julia 的类型系统

类型分为

- primitive type: 无法被分解为其它类型的组合。
- abstract type： 抽象的类型，无成员变量。
- concrete type： 类型系统中的叶子节点，可为其分配内存。

例子：

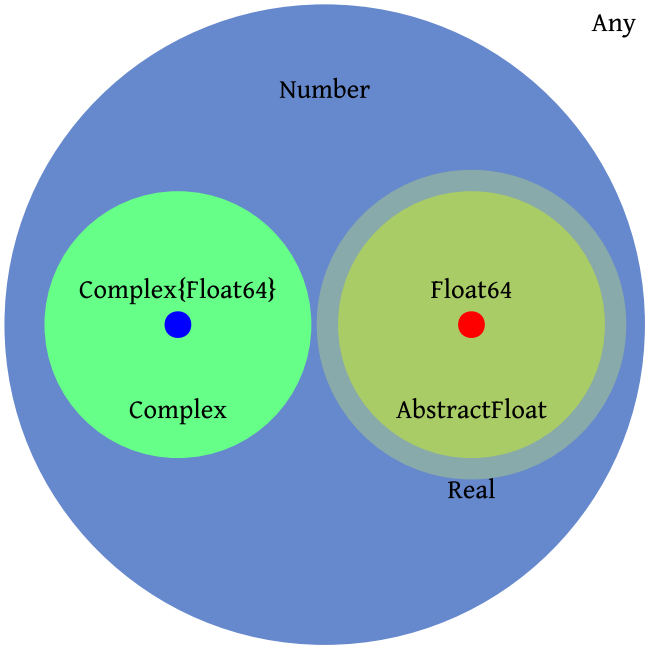
```
# the definition of an abstract type
abstract type A end
```

```
# the definition of a concrete type
struct C <: A
    member1::Float64
    member2::Int
end
```

Numbers

```
Number
├─ Complex{T<:Real}
├─ Real
│  ├─ AbstractFloat
│  │  ├─ BigFloat
│  │  ├─ Float16
│  │  ├─ Float32
│  │  └─ Float64
│  ├─ AbstractIrrational
│  │  └─ Irrational{sym}
│  ├─ FixedPointNumbers.FixedPoint{T<:Integer, f}
│  │  ├─ FixedPointNumbers.Fixed{T<:Signed, f}
│  │  └─ FixedPointNumbers.Normed{T<:Unsigned, f}
│  ├─ Integer
│  │  ├─ BitBasis.DitStr{D, N, T<:Integer}
│  │  ├─ Bool
│  │  ├─ Signed
│  │  │  ├─ BigInt
│  │  │  ├─ Int128
│  │  │  ├─ Int16
│  │  │  ├─ Int32
│  │  │  ├─ Int64
│  │  │  └─ Int8
│  │  └─ Unsigned
│  │     ├─ UInt128
│  │     ├─ UInt16
│  │     ├─ UInt32
│  │     ├─ UInt64
│  │     └─ UInt8
│  ├─ Rational{T<:Integer}
│  ├─ StatsBase.PValue
│  └─ StatsBase.TestStat
├─ TropicalNumbers.CountingTropical{T, CT}
```

```
print_type_tree(Number)
```



<: 是subtype的意思， `A <: B` 表示 A 是 B 的子集。

```
true
```

```
AbstractFloat <: Number
```

`Any` 是任意类型的 parent

```
true
```

```
Number <: Any
```

一个类型包括两个部分，类型名字和类型参数。

```
ComplexF64 (alias for Complex{Float64})
```

```
# TypeName{type parameters...}
Complex{Float64}  # 由 64 位浮点数参数化的复数
```

```
(:re, :im)
```

```
fieldnames(Complex)
```

```
true
```

```
Base.isprimitivetype(Float64)
```

```
true
```

```
Base.isabstracttype(AbstractFloat)
```

```
true
```

```
Base.isconcretetype(Complex{Float64})
```

提问： Complex 是不是 concrete type?

```
false
```

```
Base.isconcretetype(Complex)
```

```
true
```

```
Base.isconcretetype(Complex{Float64})
```

那么如何表达一个复数，它的实部和虚部都是浮点数？

```
Complex{<:AbstractFloat}
```

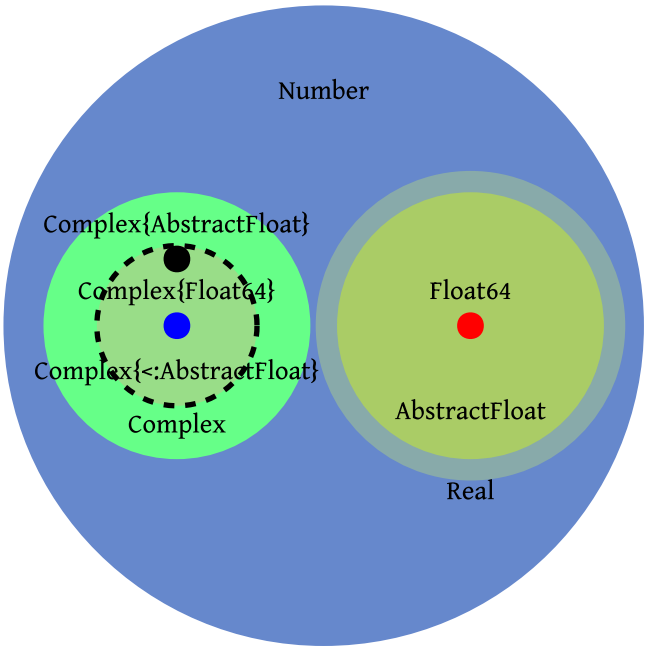
```
Complex{<:AbstractFloat}
```

```
true
```

```
Complex{Float64} <: Complex{<:AbstractFloat}
```

```
false
```

```
Complex{Float64} <: Complex{AbstractFloat}
```



猜猜是true还是false？

```
true
```

```
isconcretetype(Complex{AbstractFloat})
```

它们的区别很大！

```
vany =  []
```

```
vany = Any[]
```

```
true
```

```
vany isa Vector{Any}
```

```
true
```

```
vany isa Vector{<:Any}
```

```
["a"]
```

```
push!(vany, "a")
```

```
vfloat64 =  []
```

```
vfloat64 = Float64[]
```

```
true
```

```
vfloat64 isa Vector{<:Any}
```

```
false
```

```
vfloat64 isa Vector{Any}
```

```
MethodError: Cannot `convert` an object of type String to an object of type Float64
Closest candidates are:
convert(::Type{T}, !Matched::ColorTypes.Gray24) where T<:Real at
~/.julia/packages/ColorTypes/1dGw6/src/conversions.jl:114
convert(::Type{T}, !Matched::ColorTypes.Gray) where T<:Real at
~/.julia/packages/ColorTypes/1dGw6/src/conversions.jl:113
convert(::Type{T}, !Matched::T) where T<:Number at number.jl:6
...
```

1. **push!**(::Vector{Float64}, ::String) @ *array.jl:1057*
2. **top-level scope** @ *Local: 1* [inlined]

```
push!(vfloat64, "a")
```

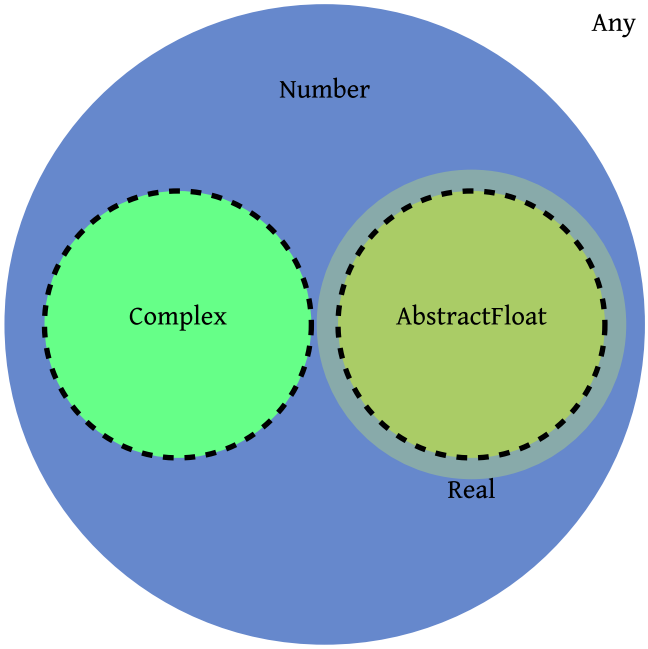
用 Union 代表两个类型的并集。

```
true
```

```
Union{AbstractFloat, Complex} <: Number
```

```
false
```

```
Union{AbstractFloat, Complex} <: Real
```



给类型起绰号

```
Union{Complex{T}, T} where T<:AbstractFloat
```

```
FloatAndComplex{T} = Union{T, Complex{T}} where T<:AbstractFloat
```

如何定义函数

```
roughly_equal (generic function with 3 methods)
```

```
begin
    # fallback
    function roughly_equal(x::Number, y::Number)
        @info "(::Number, ::Number)"
        x ≈ y   # type with \approx<TAB>
    end
    function roughly_equal(x::AbstractFloat, y::Number)
        @info "(::AbstractFloat, ::Number)"
        -10 * eps(x) < x - y < 10 * eps(x)
    end
    function roughly_equal(x::Number, y::AbstractFloat)
        @info "(::Number, ::AbstractFloat)"
        -10 * eps(y) < x - y < 10 * eps(y)
    end
end
```

3 methods for generic function **roughly_equal**:

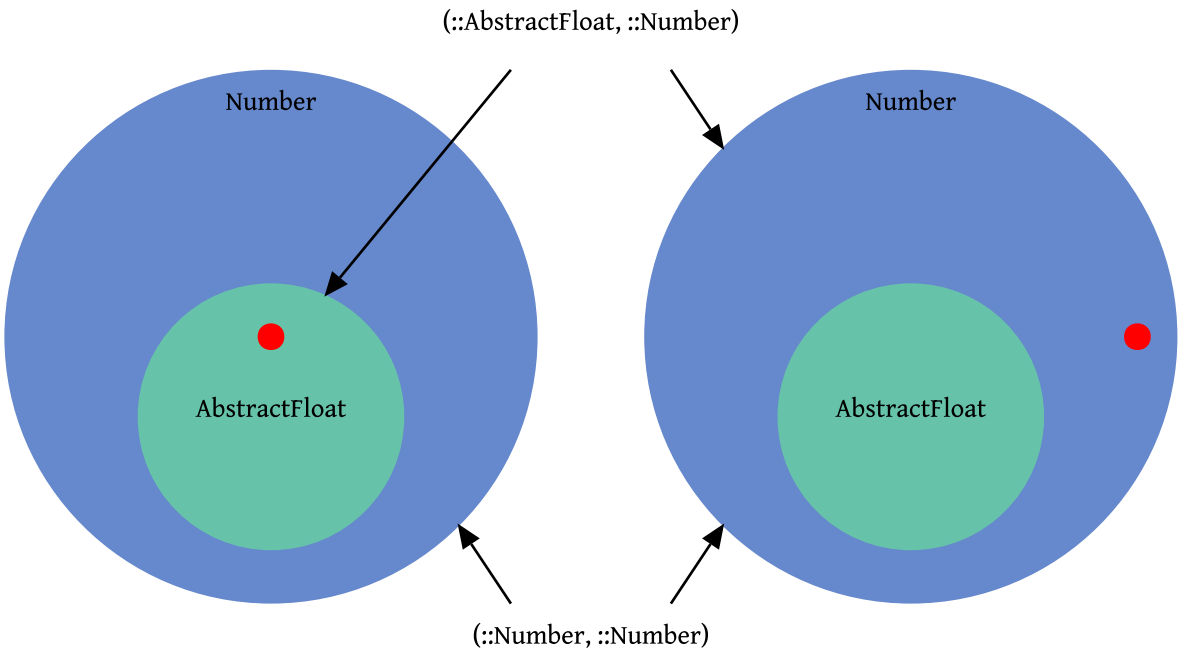
- roughly_equal(x::**AbstractFloat**, y::**Number**) in Main.var"workspace#13" at /home/leo/jcode/notebooks/julia/juliatutorial.jl#==#69fed6cc-030b-4066-a023-0bbf1637fbbc:7
- roughly_equal(x::**Number**, y::**AbstractFloat**) in Main.var"workspace#13" at /home/leo/jcode/notebooks/julia/juliatutorial.jl#==#69fed6cc-030b-4066-a023-0bbf1637fbbc:11
- roughly_equal(x::**Number**, y::**Number**) in Main.var"workspace#13" at /home/leo/jcode/notebooks/julia/juliatutorial.jl#==#69fed6cc-030b-4066-a023-0bbf1637fbbc:3

```
# `methods` is different from `methodinstances` in MethodAnalysis. It returns
method definitions rather than compiled binaries.
methods(roughly_equal)
```

```
true
```

```
roughly_equal(3.0, 3)  # case 1
```

```
(::AbstractFloat, ::Number)
```

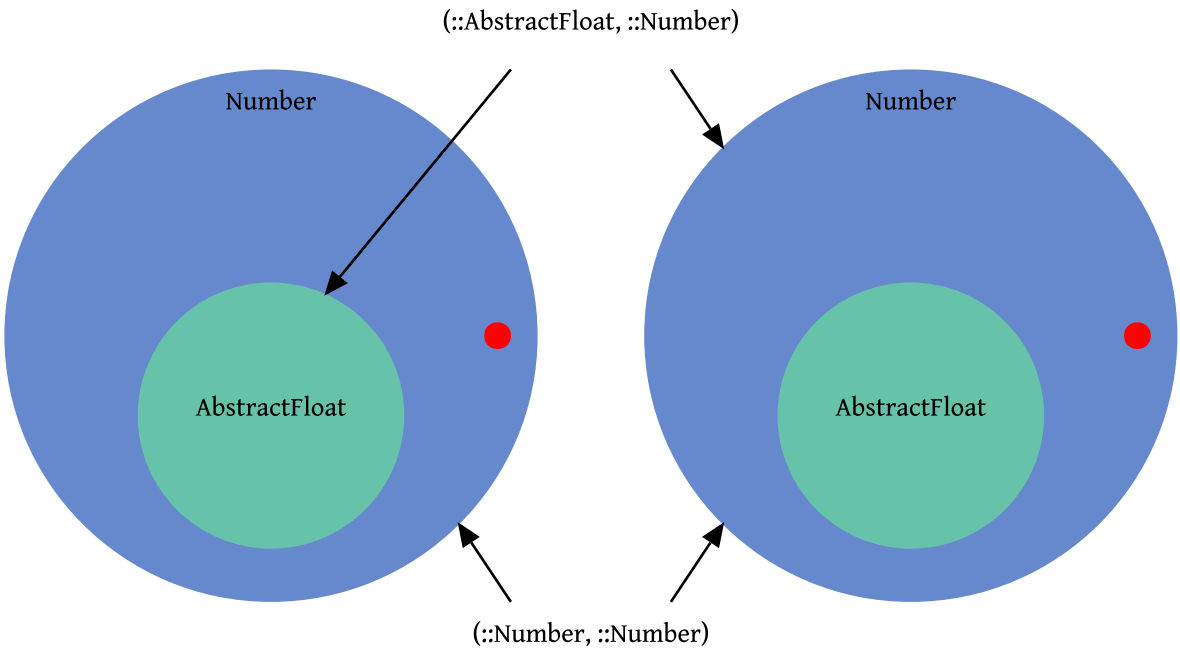


最具体的获胜

```
true
```

```
• roughly_equal(3, 3)    # case 2
```

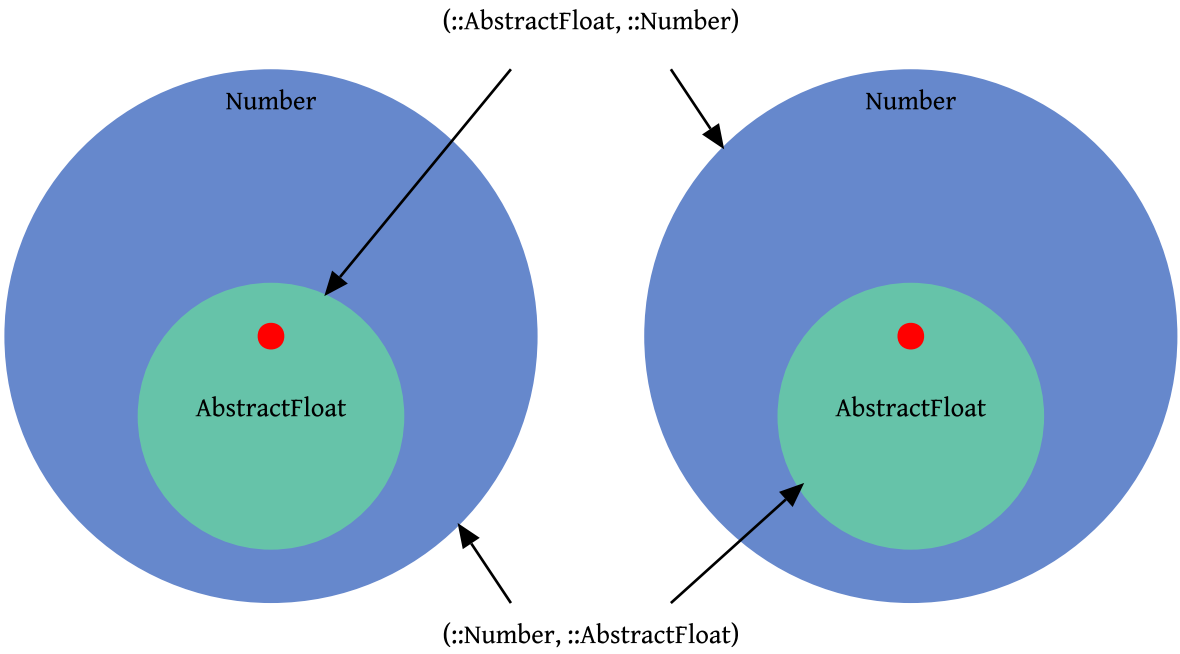
```
(::Number, ::Number)
```



```
MethodError: roughly_equal(::Float64, ::Float64) is ambiguous. Candidates:
roughly_equal(x::AbstractFloat, y::Number) in Main.var"workspace#13" at
/home/leo/jcode/notebooks/julia/juliatutorial.jl#==#69fed6cc-030b-4066-a023-
0bbf1637fbbc:7
roughly_equal(x::Number, y::AbstractFloat) in Main.var"workspace#13" at
/home/leo/jcode/notebooks/julia/juliatutorial.jl#==#69fed6cc-030b-4066-a023-
0bbf1637fbbc:11
Possible fix, define
roughly_equal(::AbstractFloat, ::AbstractFloat)
```

1. **top-level scope** @ *Local: 1* [inlined]

```
roughly_equal(3.0, 3.0)
```



有时候，难论输赢。解决方式就是定义更加具体的实现：

```
function roughly_equal(x::AbstractFloat, y::AbstractFloat)
    @info "(::AbstractFloat, ::AbstractFloat)"
    -10 * eps(y) < x - y < 10 * eps(y)
end
```

```
using MethodAnalysis
```

猜，现在 `f` 有多少个函数实例？

```
[MethodInstance for Main.var"workspace#13".roughly_equal(::Float64, ::Int64), MethodIns
```

```
methodinstances(roughly_equal)
```

让类型参数保持一致。

```
lmul (generic function with 2 methods)
```

```
begin
    function lmul(x::Complex{T1}, y::AbstractArray{<:Complex{T2}}) where {T1<:Real, T2<:Real}
        @info "(::Complex{T1}, ::AbstractArray{<:Complex{T2}}) where {T1<:Real, T2<:Real}"
        x .* y
    end
    function lmul(x::Complex{T}, y::AbstractArray{<:Complex{T}}) where T<:Real
        @info "(::Complex{T}, ::AbstractArray{<:Complex{T}}) where T<:Real"
        x .* y
    end
end
```

```
3×3 Matrix{ComplexF64}:
 0.665679-4.07168im    2.08707-2.01419im    -3.08344-0.729952im
  1.19456-0.345815im  -0.79022-0.374643im   -1.77846-4.04468im
 -0.16332-0.437878im  -1.38606-1.6596im    -0.136693-3.15202im
```

```
lmul(3.0im, randn(ComplexF64, 3, 3))
```

```
(::Complex{T}, ::AbstractArray{<:Complex{T}}) where T<:Real
```

```
3×3 Matrix{ComplexF64}:
 -0.293757+3.11911im   -1.11584+3.24027im  0.221091+0.625693im
  0.857352-0.799024im  -1.84626-1.06075im    1.1586+3.82763im
   1.67513+1.76439im   -1.26916+1.95163im   5.55834+2.85498im
```

```
lmul(3im, randn(ComplexF64, 3, 3))
```

```
(::Complex{T1}, ::AbstractArray{<:Complex{T2}}) where {T1<:Real, T2<:Real}
```

小结

- Julia 的多重派发比面向对象提供了更多的抽象的可能（指数大）。
- 可以利用巧妙的类型系统设计，在指数大的函数空间中写代码。

Julia的软件包开发

一个软件包的文件结构

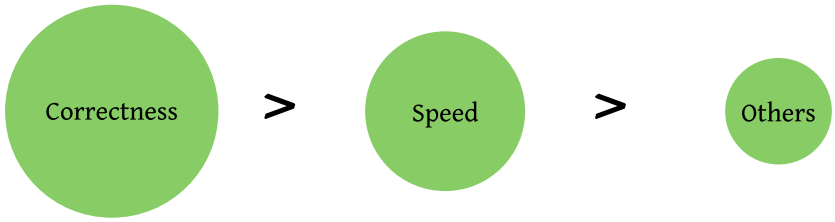
```
project_folder = "/home/leo/.julia/packages/TropicalNumbers/dCQLq"
```

```
project_folder = dirname(dirname(pathof(TropicalNumbers)))
```

```
├─ .github
│  └─ .github/workflows
│     ├─ .github/workflows/TagBot.yml
│     └─ .github/workflows/ci.yml
├─ .gitignore
├─ LICENSE
├─ Project.toml
├─ README.md
├─ docs
│  └─ docs/src
│     └─ docs/src/index.md
├─ src
│  ├─ src/TropicalNumbers.jl
│  ├─ src/counting_tropical.jl
│  └─ src/tropical.jl
└─ test
   ├─ test/counting_tropical.jl
   ├─ test/runtests.jl
   └─ test/tropical.jl
```

```
print_dir_tree(project_folder)
```

Unit Test




```
using Test
```

```
using TropicalNumbers
```

```
Test Passed
```

```
@test Tropical(3.0) + Tropical(2.0) == Tropical(3.0)
```

```
Test Passed
      Thrown: BoundsError
```

```
@test_throws BoundsError [1,2][3]
```

```
Test Broken
  Expression: 3 == 2
```

```
@test_broken 3 == 2
```

```
DefaultTestSet("Tropical Number addition", [Test Broken         ], 2, false, false, true
                                             Expression: 3 == 2
```

```
@testset "Tropical Number addition" begin
    @test Tropical(3.0) + Tropical(2.0) == Tropical(3.0)
    @test_throws BoundsError [1][2]
    @test_broken 3 == 2
end
```

```
Test Summary:             | Pass  Broken  Total  Time
Tropical Number addition  |    2       1      3  0.0s
```

```
FILE: /home/leo/.julia/packages/TropicalNumbers/dCQLq/test/runtests.jl
--------------------------------------------------------------------------------
using TropicalNumbers
using Test, Documenter

@testset "tropical" begin
    include("tropical.jl")
end

@testset "counting_tropical" begin
    include("counting_tropical.jl")
end

doctest(TropicalNumbers)
```

```
showfile(joinpath(project_folder, "test", "runtests.jl"))
```

```
   Testing TropicalNumbers
    Status `/tmp/jl_uHhT3W/Project.toml`
[e30172f5] Documenter v0.27.23
[b3a74e9c] TropicalNumbers v0.5.5
[8dfed614] Test `@stdlib/Test`
    Status `/tmp/jl_uHhT3W/Manifest.toml`
[a4c015fc] ANSIColoredPrinters v0.0.1
[ffbed154] DocStringExtensions v0.9.2
[e30172f5] Documenter v0.27.23
[b5f81e59] IOCapture v0.2.2
[682c06a0] JSON v0.21.3
[69de0a69] Parsers v2.5.1
[66db9d55] SnoopPrecompile v1.0.1
[b3a74e9c] TropicalNumbers v0.5.5
[2a0f44e3] Base64 `@stdlib/Base64`
[ade2ca70] Dates `@stdlib/Dates`
[b77e0a4c] InteractiveUtils `@stdlib/InteractiveUtils`
[76f85450] LibGit2 `@stdlib/LibGit2`
```

```
with_terminal() do
    Pkg.test("TropicalNumbers")
end
```

了解更多

版本控制与依赖

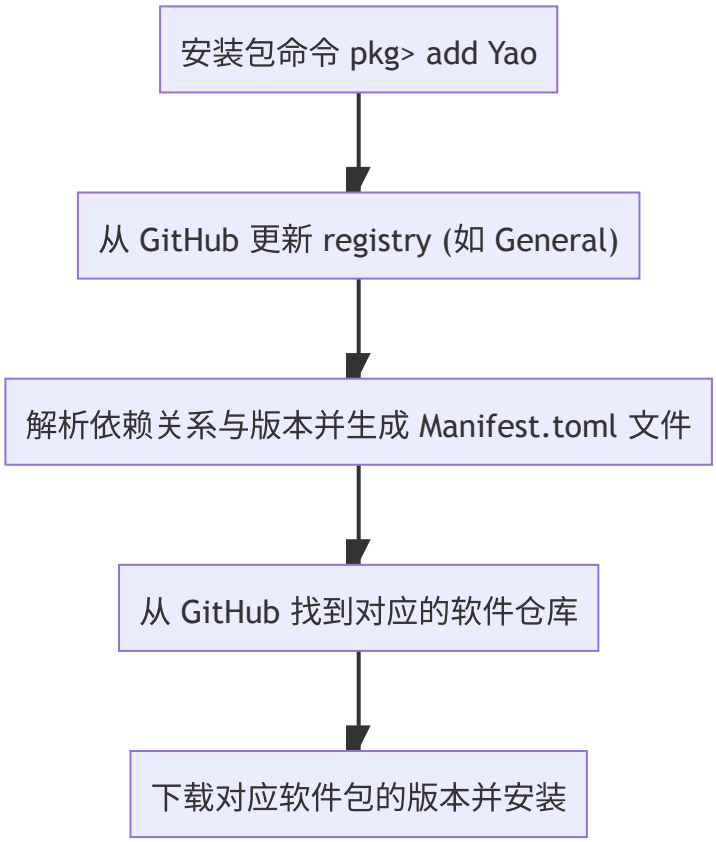
```
FILE: /home/leo/.julia/packages/Yao/8BgmW/Project.toml
--------------------------------------------------------------------------------
name = "Yao"
uuid = "5872b779-8223-5990-8dd0-5abbb0748c8c"
version = "0.8.5"

[deps]
BitBasis = "50ba71b6-fa0f-514d-ae9a-0916efc90dcf"
LinearAlgebra = "37e2e46d-f89d-539d-b4ee-838fcccc9c8e"
LuxurySparse = "d05aeea4-b7d4-55ac-b691-9e7fabb07ba2"
Reexport = "189a3867-3050-52da-a836-e630ba90ab69"
YaoAPI = "0843a435-28de-4971-9e8b-a9641b2983a8"
YaoArrayRegister = "e600142f-9330-5003-8abb-0ebd767abc51"
YaoBlocks = "418bc28f-b43b-5e0b-a6e7-61bbc1a2c1df"
YaoSym = "3b27209a-d3d6-11e9-3c0f-41eb92b2cb9d"

[compat]
BitBasis = "0.8"
LuxurySparse = "0.7"
Reexport = "1"
YaoAPI = "0.4"
YaoArrayRegister = "0.9"
YaoBlocks = "0.13"
YaoSym = "0.6"
julia = "1"

[extras]
Documenter = "e30172f5-a6a5-5a46-863b-614d45cd2de4"
LinearAlgebra = "37e2e46d-f89d-539d-b4ee-838fcccc9c8e"
Random = "9a3f8284-a2c9-5f02-9a11-845980a1fd5c"
SymEngine = "123dc426-2d89-5057-bbad-38513e3affd8"
Test = "8dfed614-e22c-5e08-85e1-65c5234f0b40"
```

```
let project_folder = dirname(dirname(pathof(Yao)))
    showfile(joinpath(project_folder, "Project.toml"))
end
```



以量子计算软件包 Yao 为例， 它的依赖关系可以非常复杂。

```
using Yao
```

```
Yao (v0.8.5)
├─ YaoArrayRegister (v0.9.3)
│  ├─ Random
│  │  ⋮
│  │
│  ├─ TupleTools (v1.3.0)
│  ├─ BitBasis (v0.8.1)
│  │  ⋮
│  │
│  ├─ LinearAlgebra
│  │  ⋮
│  │
│  ├─ YaoAPI (v0.4.3)
│  ├─ StatsBase (v0.33.21)
│  │  ⋮
```

```
# this utility is defined at the end of this notebook
print_dependency_tree(Yao; maxdepth=2)
```

PkgTemplates

案例分析: 快乐的分子

https://github.com/CodingThrust/HappyMolecules.jl

感谢

会议组织

- **刘贵欣**： 海报制作，报告人征募，活动宣传，直播
- 赵昱圣： 衣服，徽章，贴纸，证书
- 田俊，陈久宁和干则成： 会议赞助，报告人邀请和组织
- 集智小伙伴对直播与赞助的鼎力支持。

赞助商

 长期合作伙伴	 
 黄金赞助商	

手指肌肉训练

在 Julia REPL 中跟随这些视频输入以练习 Julia 的基础。 这些视频可以在我的个人网站 找到。这个notebook和相关资料将会上传到 JuliaCN org 下面的 Github repo:

https://github.com/JuliaCN/MeetUpMaterials

1. Basic types and control flow

2. Array operations

3. Data types

4. Function and multiple dispatch

5. Performance Tips

Pluto notebook 帮助函数

```
using PlutoUI
```

Table of Contents

```
TableOfContents()
```

Present

print type tree

```
using AbstractTrees
```

```
AbstractTrees.children(x::Type) = subtypes(x)
```

```
function AbstractTrees.printnode(io::IO, x::Type{T}) where T
    print(io, _typestr(T))
end
```

```
_typestr (generic function with 1 method)
```

```
_typestr(T) = T isa UnionAll ? _typestr(T.body) : T
```

```
print_type_tree (generic function with 1 method)
```

```
function print_type_tree(T; maxdepth=5)
    io = IOBuffer()
    AbstractTrees.print_tree(io, T; maxdepth)
    Text(String(take!(io)))
end
```

```
Number
├─ Complex{T<:Real}
├─ Real
│  ├─ AbstractFloat
│  │  ├─ BigFloat
│  │  ├─ Float16
│  │  ├─ Float32
│  │  └─ Float64
│  ├─ AbstractIrrational
│  │  └─ Irrational{sym}
│  ├─ FixedPointNumbers.FixedPoint{T<:Integer, f}
│  │  ├─ FixedPointNumbers.Fixed{T<:Signed, f}
│  │  └─ FixedPointNumbers.Normed{T<:Unsigned, f}
│  ├─ Integer
│  │  ├─ BitBasis.DitStr{D, N, T<:Integer}
│  │  ├─ Bool
│  │  ├─ Signed
│  │  │  ├─ BigInt
│  │  │  ├─ Int128
│  │  │  ├─ Int16
│  │  │  ├─ Int32
│  │  │  ├─ Int64
│  │  │  └─ Int8
│  │  └─ Unsigned
│  │     ├─ UInt128
│  │     ├─ UInt16
│  │     ├─ UInt32
│  │     ├─ UInt64
│  │     └─ UInt8
│  ├─ Rational{T<:Integer}
│  ├─ StatsBase.PValue
│  └─ StatsBase.TestStat
├─ TropicalNumbers.CountingTropical{T, CT}
```

```
print_type_tree(Number)
```

print directory tree

```
function AbstractTrees.children(path::Pair{String, String})
    base, fname = path
    full = joinpath(base, fname)
    isdir(full) || return Pair{String, String}[]
    return [base=>joinpath(fname, f) for f in readdir(full) if f !== ".git"]
end
```

```
function AbstractTrees.printnode(io::IO, path::Pair{String, String})
    print(io, startswith(path.second, "./") ? path.second[3:end] : path.second)
end
```

```
print_dir_tree (generic function with 1 method)
```

```
function print_dir_tree(dir; maxdepth=5)
    io = IOBuffer()
    AbstractTrees.print_tree(io, dir=>"."; maxdepth)
    Text(String(take!(io)))
end
```

```
├─ .github
│  └─ .github/workflows
│     ├─ .github/workflows/TagBot.yml
│     └─ .github/workflows/ci.yml
├─ .gitignore
├─ LICENSE
├─ Project.toml
├─ README.md
├─ benchmark
│  ├─ benchmark/benchmark-refactor.png
│  ├─ benchmark/benchmarks.jl
│  ├─ benchmark/compiletime-refactor.png
│  ├─ benchmark/compiletime_8_false_true.dat
│  ├─ benchmark/compiletime_8_false_true_master.dat
│  ├─ benchmark/compiletime_8_true_false.dat
│  ├─ benchmark/compiletime_8_true_true.dat
│  ├─ benchmark/cubenchmark.jl
│  ├─ benchmark/greedy_order.jl
│  ├─ benchmark/julia_star.jl
│  ├─ benchmark/julia_star_cpu.jl
│  ├─ benchmark/plot_benchmark.jl
│  └─ benchmark/random_contract.jl
├─ docs
│  ├─ docs/Project.toml
│  ├─ docs/einsum_define.png
│  ├─ docs/make.jl
│  └─ docs/src
│     ├─ docs/src/contractionorder.md
│     ├─ docs/src/docstrings.md
│     ├─ docs/src/extending.md
│     ├─ docs/src/implementation.md
│     ├─ docs/src/index.md
```

```
print_dir_tree("$(homedir())/.julia/dev/OMEinsum")
```

print dependency tree

```
using Pkg
```

```
pkg_registries = Pkg.Operations.Context().registries;
```

```
function AbstractTrees.children(uuid::Base.UUID)
    dep = get(Pkg.dependencies(), uuid, nothing)
    values(dep.dependencies)
end
```

```
function AbstractTrees.printnode(io::IO, uuid::Base.UUID)
    dep = get(Pkg.dependencies(), uuid, nothing)
    link = collect(Pkg.Operations.find_urls(pkg_registries, uuid))
    if length(link) > 0
        print(io, "<a href=\"$(link[1])\">$(dep.name)</a> (v$(dep.version))")
    else
        print(io, "$(dep.name)")
    end
end
```

```
print_dependency_tree (generic function with 1 method)
```

```
function print_dependency_tree(pkg; maxdepth=5)
    io = IOBuffer()
    AbstractTrees.print_tree(io, Pkg.project().dependencies[string(pkg)]; maxdepth)
    HTML("<p style='font-family: Consolas; line-height: 1.2em; max-height: 300px;'>"
     * replace(String(take!(io)), "\n"=>"<br>") * "</p>")
end

```

Mermaid diagram

```
@mermaid_str (macro with 1 method)
```

```
macro mermaid_str(str)
    return HTML("""<script
    src="https://cdn.bootcss.com/mermaid/8.14.0/mermaid.min.js"></script>
<script>
  // how to do it correctly?
  mermaid.init({
    noteMargin: 10
  }, ".someClass");
</script>

<div class="mermaid someClass">
  $str
</div>
""")
end
```

Layout

```
leftright (generic function with 1 method)
```

```
    # left right layout
    function leftright(a, b; width=600)
        HTML("""
<style>
table.nohover tr:hover td {
   background-color: white !important;
}</style>

<table width=$(width)px class="nohover" style="border:none">
<tr>
    <td>$(html(a))</td>
    <td>$(html(b))</td>
</tr></table>
""")
    end
```

Live coding

```
livecoding (generic function with 1 method)
```

```
function livecoding(src)
    HTML("""
<div></div>
<link rel="stylesheet" type="text/css" href="https://cdn.jsdelivr.net/npm/asciinema-player@3.0.1/dist/bundle/asciinema-player.css" />
<script src="https://cdn.jsdelivr.net/npm/asciinema-player@3.0.1/dist/bundle/asciinema-player.min.js"></script>
<script>
var target = currentScript.parentElement.firstElementChild;
AsciinemaPlayer.create('$src', target);
target.firstChild.firstChild.firstChild.style.background = "#000000";
target.firstChild.firstChild.firstChild.style.color = "#FFFFFF";
</script>
""")
end
```

Luxor plot utilities

```
using Luxor
```

```
begin
    function drawset!(x, y; textoffset=0, dash=false, bgcolor, r, text, opacity=1.0)
        setcolor(bgcolor)
        setopacity(opacity)
        circle(x, y, r; action=:fill)
        setopacity(1.0)
        setcolor("black")
        if dash
            setdash("dashed")
            circle(x, y, r; action=:stroke)
        end
        Luxor.text(text, x, y+textoffset; halign=:center, valign=:center)
    end
    function draw_number!(x, y; textoffset=0, dash=false)
        drawset!(x, y; textoffset, dash, bgcolor="#6688CC", r=100, text="Number")
    end
    function draw_floatandcomplex!(x, y; textoffset=0, dash=false)
        drawset!(x, y; textoffset, dash, bgcolor="#AACC66", r=50,
        text="FloatAndComplex")
    end
    function draw_float!(x, y; textoffset=0, dash=false)
        drawset!(x, y; textoffset, dash, bgcolor="#66FF88", r=50,
        text="AbstractFloat", opacity=0.5)
    end
    function ring3!(x, y)
        draw_number!(x, y; textoffset=-85)
        #draw_floatandcomplex!(x, y-30; textoffset=0)
        draw_float!(x, y+30; textoffset=0)
    end
    @drawsvg begin
        ring3!(0, 0)
    end 300 300
end;
```

display a file

```
showfile (generic function with 1 method)
```

```
function showfile(filename)
    Text("FILE: $filename\n"* ("-"^80) * "\n" * read(filename, String) * "\n")
end
```