Julia Fast scientific language

Generic function name: area → `call with args` → **Dispatch** all argument types → `first call` → **Method** specialized, compiled → `cached` → **Fast native code** reused after

CONCEPTS

Julia compiles each method on first call, so the first run is slow and later calls are fast. Keep one REPL open.

Functions have many methods; the one chosen depends on the types of all arguments (multiple dispatch).

Arrays are 1-indexed, column-major, and a function ending in ! mutates its first argument.

COMMAND LINE

`julia`
Start the interactive REPL

`julia <file.jl> <args>`
Run a script; args land in ARGS

`julia -e '<expr>'`
Evaluate an expression and exit

`julia -E '<expr>'`
Evaluate and print the result

`julia --project=<dir>`
Activate the environment in a folder

`julia -t <n|auto>`
Set the number of threads

COMMAND LINE (cont.)

`julia -i <file.jl>`
Run a file, then stay in the REPL

REPL

`?`
Help mode: docs for a name

`]`
Package mode

`;`
Shell mode: run a shell command

`Backspace`
Leave a special mode

`Ctrl+D / exit()`
Quit Julia

`Ctrl+C`
Interrupt running code

`@time <expr>`
Time an expression and show allocations

`include("<file.jl>")`
Evaluate a file in this session

`using <Pkg>`
Load a package's exports

PACKAGES

`add <pkg>`
Add a package to the environment

PACKAGES (cont.)

`rm <pkg>`
Remove a package

`status`
Show the environment (alias st)

`up`
Update packages

`activate <dir>`
Switch to the environment in a folder

`instantiate`
Install what Manifest.toml lists

`test`
Run the project's tests

BASICS

`x = 3; y = x^2`
Assign and exponentiate

`println("x = $x")`
Print with interpolation

`if a < b ... elseif ... else ... end`
Conditional

`for i in 1:10 ... end`
Loop over a range

`function f(x, y=2; k=1) ... end`
Function with default and keyword args

BASICS (cont.)

`f(x) = x^2`
Short function form

`x → x^2`
Anonymous function

`Dict("a" ⇒ 1)`
Dictionary

ARRAYS

`[1, 2, 3]`
Vector literal

`[1 2; 3 4]`
Matrix literal

`A[i, j]`
Element at row i, column j

`A[2:end, :]`
Slice; end is the last index

`push!(v, x)`
Append to a vector

`length(v) / size(A)`
Element count / dimensions

`[f(x) for x in v if <cond>]`
Comprehension

`f.(v)`
Broadcast f over elements

`v .+ 1`
Elementwise operator

`map(f, v) / filter(f, v)`
Apply / keep matching

ARRAYS (cont.)

`v[v .> 0]`
Select by boolean mask

TYPES & DISPATCH

`struct P x::Float64; y::Float64 end`
Immutable composite type

`mutable struct C n::Int end`
Mutable composite type

`area(s::Circle) = pi * s.r^2`
Method for one type

`abstract type Shape end`
Abstract supertype

`struct Circle <: Shape ... end`
Declare a subtype

MACROS & TOOLING

`@time f(x)`
Time and allocation report

`@btime f($x)`
Accurate benchmark (BenchmarkTools)

HELP

`?<name>`
Docs for a function or type