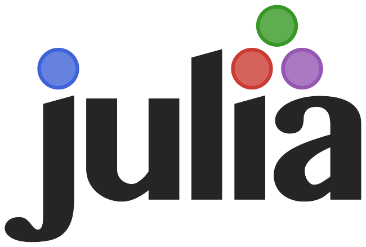
Getting Started with Julia

By Erik Stryshak



|  |  |
| --- | --- |
| Website | https://julialang.org |
| Documentation | https://docs.julialang.org/en/stable/ |
| Packages | https://pkg.julialang.org |
| Download | https://julialang.org/downloads/ |

Julia is a programming language whose primary purpose is in numerical analysis and scientific computing. It is a high-level language that strives to approach the performance in C. The syntax is like that of python and it contains many of the mathematical functions found in MATLAB, R, and the NumPy package. The typing is dynamic and non-strict.

|  |  |  |  |
| --- | --- | --- | --- |
| Variable/Collection | | Syntax | Notes |
| String | s = “string" | |  |
| Int | i = 6 | |  |
| Boolean | b1 = true, b2 = false | |  |
| Float | f = 6.3 | |  |
| Array 1 | a1 = Array{Type}(Dimensions) | |  |
| Array 2 | a2 = Int64[val1, val2, val3, …] | | Using commas creates a column vector. Omitting commas creates a row vector |
| Array 3 | a3 = [val1, val2, val3] | | If type is not provided, then the type is “Any” |
| Array 4 | a4 = [val1 val2 ; val3 val4] | | Creates a 2x2 array |
| Dictionary 1 | d1 = Dict(key=>val, key=>val) | |  |
| Dictionary 2 | d2 = Dict([(key,val),(key,val)]) | |  |
| Tuple | t = (val1, val2, …) | | Tuples are immutable |

|  |  |  |
| --- | --- | --- |
| Action | Code | Notes |
| Hello World | println(“Hello World”) |  |
| Function Syntax 1 | function foo(arg)  …code…  end | Function that takes in argument arg which can be of any type |
| Function Syntax 2 | function foo(arg::Int64)  …code…  return arg + 1  end | Function that takes in an argument that must be an Int64, and returns a value |
| Function Syntax 3 | foo(arg) = ..code.. | Equivalent to Function Syntax1 |
| Function Call | foo(3) | Calls function foo and passes 3 |
| Object Definition | struct foo  var\_1  var\_2::Int64  end | Creates an object foo with two parameters. var\_1 can be any type, while var\_2 must be an Int64 |
| Object Creation | f = foo(“bar”, 5) | Creates a foo object f |
| Access Object Data | f.var\_1 | Gets the var\_1 variable of f |
| Add a Package | Pkg.add(“Package”) |  |
| Include a Package | Using Package |  |
| Comment | # a comment |  |
| Error Checking | arr = [1, 2, 3, 4]  try  println(arr[0])  catch e  println(e)  end | In Julia, array indexing begins at 1, so the statement “println(arr[0])” will throw a bounds error |
| Conditional | if var > 10  …  elseif var < 10  …  else  …  end |  |
| Looping | arr = [1, 2, 3, 4]  For number in arr  println(“num: $num”)  end | In println statement, the $ operator performs string interpolation of the variable |