Julia: Functions, Modules, Projects and Packages

Bill McLean

School Colloquium
January, 2022

Functions

Consider a simple Julia function

$$f(x, y) = 2x + y.$$

- If I call f with integer arguments, Julia compiles and runs a version of this function that works when x and y are integers, and returns an integer result.
- The compiled code is cached so it can be re-used as many times as needed.

Functions

Consider a simple Julia function

$$f(x, y) = 2x + y.$$

- If I call f with integer arguments, Julia compiles and runs a version of this function that works when x and y are integers, and returns an integer result.
- The compiled code is cached so it can be re-used as many times as needed.
- If I then call f with floating-point arguments, Julia compiles and runs a different version that works when x and y are floating-point numbers, and returns a floating-point result.

Functions

Consider a simple Julia function

$$f(x, y) = 2x + y.$$

- If I call f with integer arguments, Julia compiles and runs a version of this function that works when x and y are integers, and returns an integer result.
- The compiled code is cached so it can be re-used as many times as needed.
- If I then call f with floating-point arguments, Julia compiles and runs a different version that works when x and y are floating-point numbers, and returns a floating-point result.
- So the calls f(1,2), f(1.0,2.0), f(1.0,2) and f(1,2.0) all use different compiled versions of f.

Compilation

- Julia relies on the LLVM compiler framework (like Clang).

Compilation

- Julia relies on the LLVM compiler framework (like Clang).
- You can look at the assembler generated by Julia using the code_native function or the associated @code_native macro.
- E.g., try code_native(f, (Int64, Int64)) or @code_native f(2, 3).

Compilation

- Julia relies on the LLVM compiler framework (like Clang).
- You can look at the assembler generated by Julia using the code_native function or the associated @code_native macro.
- E.g., try code_native(f, (Int64, Int64)) or @code_native f(2, 3).
- Similarly, code_llvm shows LLVM intermediate representation.
- Unlike a normal compiler, the compiled code is not available to the user as an object file.
- Thus, if you restart Julia then your function has to be recompiled.

Compilation

- Julia relies on the LLVM compiler framework (like Clang).
- You can look at the assembler generated by Julia using the code_native function or the associated @code_native macro.
- E.g., try code_native(f, (Int64, Int64)) or @code_native f(2, 3).
- Similarly, code_llvm shows LLVM intermediate representation.
- Unlike a normal compiler, the compiled code is not available to the user as an object file.
- Thus, if you restart Julia then your function has to be recompiled.
- For large projects, functions should be organised into modules. These can be precompiled and stored in $HOME/.julia.
- Julia relies on type inference. Any type instability can harm performance (code_warntype). Read the "Performance Tips" in the manual.

Methods

- Julia functions are generic. A given f can have multiple definitions, called methods. E.g., consider methods(eigen).
- Multiple dispatch refers to the system Julia uses to select the appropriate method for a given function call.

Methods

- Julia functions are generic. A given f can have multiple definitions, called methods. E.g., consider methods(eigen).
- Multiple dispatch refers to the system Julia uses to select the appropriate method for a given function call.
- Our f is an example of a generic function with 1 method. Julia will accept x and y of any type such that the expression 2x+y makes sense. E.g., x and y could be matrices of the same size.
- Julia does a good job of finding the most specific method for a given function call.

Methods

- Julia functions are generic. A given f can have multiple definitions, called methods. E.g., consider methods(eigen).
- Multiple dispatch refers to the system Julia uses to select the appropriate method for a given function call.
- Our f is an example of a generic function with 1 method. Julia will accept x and y of any type such that the expression 2x+y makes sense. E.g., x and y could be matrices of the same size.
- Julia does a good job of finding the most specific method for a given function call.
- Even if a function has only 1 method, type assertions can be useful for constraining the allowed types.

```
function custom_gauss_rule(lo::T, hi::T,
           a::Vector{T}, b::Vector{T}
           ) where {T<:AbstractFloat}
```

Modules

- Modules are a convenient mechanism for grouping related type definitions and functions in a way that makes them usable by any program.

Modules

- Modules are a convenient mechanism for grouping related type definitions and functions in a way that makes them usable by any program.

Modules

- Modules are a convenient mechanism for grouping related type definitions and functions in a way that makes them usable by any program.
- A Julia module cannot have an entry point.
- Each module provides a restricted name space and Julia provides disambiguation mechanisms to sort out name clashes. E.g., CSV.read or alternatively

  ```
  import CSV: read as readcsv
  ```

Modules

- Modules are a convenient mechanism for grouping related type definitions and functions in a way that makes them usable by any program.
- A Julia module cannot have an entry point.
- Each module provides a restricted name space and Julia provides disambiguation mechanisms to sort out name clashes. E.g., CSV.read or alternatively

  ```
  import CSV: read as readcsv
  ```
- The built-in variable LOAD_PATH (or the environment variable JULIA_LOAD_PATH) tells Julia which directories to search for modules. Alternatively, use a project environment (see below).

Modules

- Modules are a convenient mechanism for grouping related type definitions and functions in a way that makes them usable by any program.
- A Julia module cannot have an entry point.
- Each module provides a restricted name space and Julia provides disambiguation mechanisms to sort out name clashes. E.g., CSV.read or alternatively

  ```
  import CSV: read as readcsv
  ```
- The built-in variable LOAD_PATH (or the environment variable JULIA_LOAD_PATH) tells Julia which directories to search for modules. Alternatively, use a project environment (see below).
- The Revise package is useful when developing modules.
- Organise large modules into submodules.

Projects

- Typically organise a project into directories src/, test/, docs/, scripts/, etc.
- The pkg command generate is helpful when starting a new project.

Projects

- Typically organise a project into directories src/, test/, docs/, scripts/, etc.
- The pkg command generate is helpful when starting a new project.
- The module in src/ directory holds functions that can be used in multiple scripts.
- Typically host code on github.
- The Test package provides support for unit testing (with an automated build status on github).

Projects

- Typically organise a project into directories src/, test/, docs/, scripts/, etc.
- The pkg command generate is helpful when starting a new project.
- The module in src/ directory holds functions that can be used in multiple scripts.
- Typically host code on github.
- The Test package provides support for unit testing (with an automated build status on github).
- Ideal way to share code associated with a paper.

Packages

- A package is just a project intended for use in other projects. It has to conform strictly to the recommended practices for projects.
- If hosted online, it can be installed by the pkg command add *url*.
- If a package is registered, then just do add *package name* instead.
- Registered packages can be downloaded from the nearest mirror site. This infrastructure is funded by Julia Computing.

Packages

- A package is just a project intended for use in other projects. It has to conform strictly to the recommended practices for projects.
- If hosted online, it can be installed by the pkg command add *url*.
- If a package is registered, then just do add *package name* instead.
- Registered packages can be downloaded from the nearest mirror site. This infrastructure is funded by Julia Computing.
- Programming language adoption often driven by available packages as much as (or more than) features.
- No modern language could be competitive without a package manager.

Dependency Hell

- Any non-trivial project depends on some modules external to the project.
- More challenging projects often rely on modules from third-party packages.

Dependency Hell

- Any non-trivial project depends on some modules external to the project.
- More challenging projects often rely on modules from third-party packages.
- What if I want to run (or modify) code I wrote several years ago?
  - Good news: Julia itself (including its standard library) is backwards compatible to version 1.0. You can also install multiple Julia versions on the same computer without problems.
  - Bad news: third-party packages are another matter (especially the cutting-edge ones under rapid development).

Dependency Hell

- Any non-trivial project depends on some modules external to the project.
- More challenging projects often rely on modules from third-party packages.
- What if I want to run (or modify) code I wrote several years ago?
  - Good news: Julia itself (including its standard library) is backwards compatible to version 1.0. You can also install multiple Julia versions on the same computer without problems.
  - Bad news: third-party packages are another matter (especially the cutting-edge ones under rapid development).
- I want to use PkgA and PkgB, both depending on PkgC. What if PkgA works only with an old version of PkgC, but PkgB works only with a new version?

Environments

- Julia's package management system supports the use of project environments.
- Similar to virtual environments in Python.

Environments

- Julia's package management system supports the use of project environments.
- Similar to virtual environments in Python.
- Workflow uses pkg commands activate and add.
- File Project.toml keeps a list of all direct dependencies and their versions.
- File Manifest.toml holds exhaustive data on all dependencies, both direct and indirect.

Environments

- Julia's package management system supports the use of project environments.
- Similar to virtual environments in Python.
- Workflow uses pkg commands activate and add.
- File Project.toml keeps a list of all direct dependencies and their versions.
- File Manifest.toml holds exhaustive data on all dependencies, both direct and indirect.
- The pkg command instantiate will download and precompile all dependencies for the current project, for the specific versions recorded in the manifest file.

Environments

- Julia's package management system supports the use of project environments.
- Similar to virtual environments in Python.
- Workflow uses pkg commands activate and add.
- File Project.toml keeps a list of all direct dependencies and their versions.
- File Manifest.toml holds exhaustive data on all dependencies, both direct and indirect.
- The pkg command instantiate will download and precompile all dependencies for the current project, for the specific versions recorded in the manifest file.
- Facilitates reproducible research.