Introduction to Julia

HA Van Thao
Faculty of Math & Computer Science, HCMUS

What is Julia?

- a new programming language for scientic computing
- developed by a group mostly from MIT
- MIT licensed: free and open source
- convenient syntax for building math constructs like vectors, matrices, etc.
- good performance, approaching that of statically-compiled languages like C
- designed for parallelism and distributed computation

Installation

- The best option is probably to install the current release from the download page http://julialang.org/downloads/
- You should now be able to start Julia either by
  - navigating to Julia through your menus or desktop icons
  - opening a terminal and typing `julia`
- *Juno IDE*: Please see the Juno website for setup instructions
  - getting Atom. In Atom, install `uber-juno` package
  - Atom will then set up Juno for you, installing the required Atom and Julia packages
- Web-based interactive IJulia Notebook. JuliaBox provides a way to run IJulia notebooks in your browser.

Setting up Julia Environment

- Installing Packages.
  - Julia includes many useful tools.
  - `julia> Pkg.add("DataFrames")`
  - `julia> using DataFrames`
  - `julia> df = DataFrame(x1=[1, 2], x2=["foo", "bar"])`
- Running Julia Scripts.
  - Suppose we have a script called `test_script.jl` that we wish to run
  - Then you can run it directly from within Julia by typing `include("test_script.jl")`
  - Find out what your present working directory by typing `pwd()`
  - Change Julia's present working directory to the location of the script: `cd("/Users/hvthao")`
- Plots
  - `julia> Pkg.add("Plots")`
  - `julia> using Plots`
  - `julia> plot(sin, -2pi, pi, label="sine function")`

Data in Julia

- Integers (Int8, UInt8, ..): 1234
- Floating-points (Float16, ..): 1.23, 3.77e-7
- Booleans: true or false
- Strings: "Hello, world!"
- Characters: 'x'
- Complex Numbers: 1 + 2im
- Rational Numbers: 2//3

Setting variables

- assignments use the = operator
  - `value = 5.0`
  - `name = "Bob"`

Basic arithmetic and mathematical functions

- +, -, *, / operators
  - `a = 4.0`
  - `b = 7.0`
  - `c = (b - a) * (b + a) / a`
- exponentiate using ^
  - `a = 2 ^ 10`
- all the usual math functions, like exp, sin, . . .
  - `result = exp(-2.33) * cos(22 * 180 / pi)`

Boolean expressions

- evaluate to true or false
- use the ==, !=, <, >, <=, >= operators
  - `value = 4`
  - `value == 4`
  - `value == "4"`
  - `value > 9.0`
  - `value <= 5.3`
- flip the value of a boolean expression using !
  - !(value == "4")
- combine boolean expressions using && and ||
  - `(value == 4) && (value == "4")`
  - `(value == 4) || (value == "4")`

Ha Van Thao (hvthao@gmail.com)

If/else statements

- test if a boolean expression is true or false and run dierent code in each case

```
if (value < 5)
  value = 10
else
  value = 20
end
```

- can split the code into more than two cases

```
if (value < 5)
  value = 10
elseif (value == 5)
  value = 15
else
  value = 20
end
```

Ranges

- create a sequence of numbers using `:`
- the sequence includes the endpoints
  - `1:5`
- optional middle argument gives increment (default is 1)
  - `1:0.1:10`
- to view a range, call `collect(1:0.1:5)`

Lists

- create a numbered list of objects of dierent types using []
  - `my_list = ["a", 1, -0.76]`
- can access the ith element of the list using [i]
  - `my_list[2] + my_list[3]`
- unlike many other programming languages, Julia indexes start at 1
  - `my_list[1]` # first element of the list
  - `my_list[0]` # issues an error
- access from the end of a list using end
  - `my_list[end]` # last element of the list
  - `my_list[end - 1]` # second to last element
- use length to find how long the list is
  - `length(my_list)`

For loops

- executes a code block multiple times
- most common construction involves looping over a range

```
value = 0
for i in 1:10
  value += i # short for value = value + i
end
```

- or you can loop over a list

```
value = 0
my_list = [1, 2, 3, 4, 5]
for i in my_list
  value += i
end
```

Functions

- a chunk of code that can be run over an over, e.g.,
  - `println("Hello, World!")`
  - `println("How are you doing?")`
  - `println(49876)`
- functions can take arguments, e.g., println prints its argument
- functions can return a value, which can be stored in a variable
  - `length_of_list = length(my_list)`
- functions can have a side eect (i.e., do something), e.g., `println` prints something to the screen

Some important functions

- quit Julia: `quit()`
- print information about a function: `help(sin)`
- generate a random number between 0 and 1: `rand()`

Preferences

- http://julialang.org