Introduction to Julia

David Zeng Keegan Go Stephen Boyd

EE103
Stanford University

September 26, 2014

What is Julia?

a new programming language for scientific computing

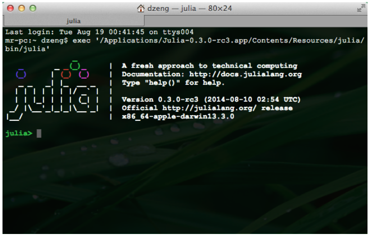
- developed by a group mostly from MIT
- fully open source, *i.e.*, free
- convenient syntax for building math constructs like vectors, matrices, etc.
- super fast

Installing Julia

- download Julia v0.3.1 from `http://julialang.org/downloads/`
- make sure to avoid v0.2.1 and v0.1.2
- follow the on-screen instructions to install

The Julia terminal

- environment to run snippets of Julia code
- open the Julia application after installation
- you should see something like this:



```
Last login: Tue Aug 19 00:41:45 on ttys004
mr-pc:~ dzeng$ exec '/Applications/Julia-0.3.0-rc3.app/Contents/Resources/julia/
bin/julia'
               _
   _       _ _(_)_     |  A fresh approach to technical computing
  (_)     | (_) (_)    |  Documentation: http://docs.julialang.org
   _ _   _| |_  __ _   |  Type "help()" for help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 0.3.0-rc3 (2014-08-10 02:54 UTC)
 _/ |\__'_|_|_|\__'_|  |  Official http://julialang.org/ release
|__/                   |  x86_64-apple-darwin13.3.0

julia>
```

- remaining slides will focus on examples using the terminal

Basic types

- integers: `Int64`, *e.g.*, `-135`
- real numbers: `Float64`, *e.g.*, `1.23`, `3.77e-7`
- to force Julia to interpret an integer as a real number, use `2.0`
- booleans: `Bool`, `true` or `false`
- strings: `ASCIIString`, *e.g.*, `"Hello, world!"`

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")
```

If/else statements

- test if a boolean expression is `true` or `false` and run different 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)

  ```
  1:2:10
  ```

Lists

- create a numbered list of objects of different types using $\{\}$, *e.g.*,

```
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 effect (*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()`

Suppressing output

- running a command in the Julia terminal will automatically print its output
- turn off output by ending a line with ;

```
value = 3
value = 3;
```

Running Julia scripts

- the Julia terminal can run files with Julia code
- use the command `pwd()` to see which folder you are currently in
- use the command `cd` to change folders

  ```
  cd("Documents/ee103")
  ```
- run a file using the `include` command

  ```
  include("testfile.jl")
  ```

Packages

- code contributed by the community that is not part of the basic installation, *e.g.*, plotting
- install an official Julia package with the `Pkg.add` function, *e.g.*, to install the plotting package Gadfly, simply specify the name

  ```
  Pkg.add("Gadfly")
  ```
- update all installed packages with `Pkg.update`

  ```
  Pkg.update()
  ```
- to use the code in a package, use the `using` command

  ```
  using Gadfly
  ```
- try plotting some points!

  ```
  x_values = 0:0.1:10
  plot(x=x_values, y=sin(x_values), Geom.point)
  ```

Other resources

- these slides only scratch the surface of the features of Julia
- more tutorials can be found here:
  `http://julialang.org/teaching/`