

Julia Programming for Operations Research 2/e

Changhyun Kwon

Julia Programming for Operations Research

<https://www.chkwon.net/julia>

Second Edition

Published by Changhyun Kwon

Copyright © 2019 by Changhyun Kwon

All Rights Reserved.

Paper and electronic versions are available at <https://www.chkwon.net/julia> and can be purchased at [Amazon](#), [Google Play](#), etc.

Cover Design by Joo Yeon Woo / www.spacekite.net
Cat Drawing by Bomin Kwon

version 2021/08/16 23:08:00

Contents

1	Introduction and Installation	1
1.1	What is Julia and Why Julia?	2
1.2	Installing Julia	4
1.2.1	Installing Julia in Windows	4
1.2.2	Installing Julia in macOS	10
1.2.3	Running Julia Scripts	15
1.2.4	Installing Gurobi	15
1.2.5	Installing CPLEX	16
1.3	Installing IJulia	18
1.4	Package Management	21
1.5	Help	25
2	Simple Linear Optimization	29
2.1	Linear Programming (LP) Problems	30
2.2	Alternative Ways of Writing LP Problems	34
2.3	Yet Another Way of Writing LP Problems	37
2.4	Mixed Integer Linear Programming (MILP) Problems	38

3 Basics of the Julia Language	41
3.1 Vector, Matrix, and Array	41
3.2 Tuple	47
3.3 Indices and Ranges	48
3.4 Printing Messages	51
3.5 Collection, Dictionary, and For-Loop	54
3.6 Function	57
3.7 Scope of Variables	59
3.8 Random Number Generation	63
3.9 File Input/Output	67
3.10 Plotting	72
3.10.1 The PyPlot Package	72
3.10.2 Avoiding Type-3 Fonts in PyPlot	77
4 Selected Topics in Numerical Methods	79
4.1 Curve Fitting	79
4.2 Numerical Differentiation	84
4.3 Numerical Integration	87
4.4 Automatic Differentiation	91
5 The Simplex Method	95
5.1 A Brief Description of the Simplex Method	95
5.2 Searching All Basic Feasible Solutions	98
5.3 Using the JuMP Package	104
5.4 Pivoting in Tableau Form	105
5.5 Implementing the Simplex Method	107
5.5.1 <code>initialize(c, A, b)</code>	109
5.5.2 <code>is_optimal(tableau)</code>	111
5.5.3 <code>pivoting!(tableau)</code>	112
5.5.4 Creating a Module	116
5.6 Next Steps	122
6 Network Optimization Problems	123
6.1 The Minimal-Cost Network-Flow Problem	123
6.2 The Transportation Problem	133
6.3 The Shortest Path Problem	139

6.4	Implementing Dijkstra's Algorithm	144
7	Interior Point Methods	151
7.1	The Affine Scaling Algorithm	151
7.2	The Primal Path Following Algorithm	157
7.3	Remarks	162
8	Nonlinear Optimization Problems	165
8.1	Unconstrained Optimization	165
8.1.1	Line Search	165
8.1.2	Unconstrained Optimization	167
8.1.3	Box-constrained Optimization	168
8.2	Nonlinear Optimization	169
8.3	Other Solvers	170
8.4	Mixed Integer Nonlinear Programming	175
9	Monte Carlo Methods	177
9.1	Probability Distributions	177
9.2	Randomized Linear Program	179
9.3	Estimating the Number of Simple Paths	186
10	Lagrangian Relaxation	197
10.1	Introduction	197
10.1.1	Lower and Upper Bounds	198
10.1.2	Subgradient Optimization	200
10.1.3	Summary	200
10.2	The p -Median Problem	201
10.2.1	Reading the Data File	202
10.2.2	Solving the p -Median Problem Optimally	204
10.2.3	Lagrangian Relaxation	205
10.2.4	Finding Lower Bounds	206
10.2.5	Finding Upper Bounds	210
10.2.6	Updating the Lagrangian Multiplier	212

11 Complementarity Problems	225
11.1 Linear Complementarity Problems (LCP)	225
11.2 Nonlinear Complementarity Problems (NCP)	233
11.3 Mixed Complementarity Problems (MCP)	237
12 Parameters in Optimization Solvers	239
12.1 Setting CPU Time Limit	239
12.2 Setting the Optimality Gap Tolerance	240
12.3 Warmstart	241
12.4 Big- M and Integrality Tolerance	242
12.5 Turning off the Solver Output	244
12.6 Other Solver Parameters	244
Index	247

Preface

The main motivation of writing this book was to help myself. I am a professor in the field of operations research, and my daily activities involve building models of mathematical optimization, developing algorithms for solving the problems, implementing those algorithms using computer programming languages, experimenting with data, etc. Three languages are involved: human language, mathematical language, and computer language. My students and I need to go over three different languages. We need “translation” among the three languages.

When my students seek help on the tasks of “translation,” I often provide them with my prior translation as an example or find online resources that may be helpful to them. If students have proper background with proper mathematical education, sufficient computer programming experience, and good understanding of how numerical computing works, students can learn easier and my daily tasks in research and education would go smoothly.

To my frustration, however, many graduate students in operations research take long time to learn how to “translate.” This book is to help them and help me to help them.

I’m neither a computer scientist nor a software engineer. Therefore, this book does not teach the best translation. Instead, I’ll try to teach how one can finish some common tasks necessary in research and development works arising in the field of operations research and management science. It will be just one translation, not

the best for sure. But after reading this book, readers will certainly be able to get things done, one way or the other.

What this book teaches

This book is *neither* a textbook in numerical methods, a comprehensive introductory book to Julia programming, a textbook on numerical optimization, a complete manual of optimization solvers, *nor* an introductory book to computational science and engineering—it is a little bit of all.

This book will first teach how to install the Julia Language itself. This book teaches a little bit of syntax and standard libraries of Julia, a little bit of programming skills using Julia, a little bit of numerical methods, a little bit of optimization modeling, a little bit of Monte Carlo methods, a little bit of algorithms, and a little bit of optimization solvers.

This book by no means is complete and cannot serve as a standalone textbook for any of the above-mentioned topics. In my opinion, it is best to use this book along with other major textbooks or reference books in operations research and management science. This book assumes that readers are already familiar with topics in optimization theory and algorithms or are willing to learn by themselves from other references. Of course, I provide the best references of my knowledge to each topic.

After reading this book and some coding exercises, readers should be able to search and read many other technical documents available online. This book will just help the first step to computing in operations research and management science. This book is literally a primer on computing.

How this book can be used

This book will certainly help graduate students (and their advisors) for tasks in their research. First year graduate students may use this book as a *tutorial* that guides them to various optimization solvers and algorithms available. This book will also be a *companion* through their graduate study. While students take various courses during their graduate study, this book will be always a good starting point to learn how to solve certain optimization problems and implement algorithms they learned. Eventually, this book can be a helpful *reference* for their thesis research.

Advanced graduate students may use this book as a *reference*. For example, when they need to implement a Lagrangian relaxation method for their own problem, they can refer to a chapter in this book to see how I did it and learn how they may be able to do it.

It is also my hope that this book can be used for courses in operations research, analytics, linear programming, nonlinear programming, numerical optimization, network optimization, management science, and transportation engineering, as a *supplementary textbook*. If there is a short course with 1 or 2 credit hours for teaching numerical methods and computing tools in operations research and management science, this book can be *primary or secondary textbook*, depending on the instructor's main focus.

Notes to advanced programmers

If you are already familiar with computing and at least one computer programming language, I don't think this book will have much value for you. There are many resources available on the web, and you will be able to learn about the Julia Language and catch up with the state-of-the-art easily. If you want to learn and catch up even faster with much less troubles, this book can be helpful.

I had some experiences with MATLAB and Java before learning Julia. Learning Julia was not very difficult, but exciting and fun. I just needed a good "excuse" to learn and use Julia. Check what my excuse was in the first chapter.

Acknowledgment

I sincerely appreciate all the efforts from Julia developers. The Julia Language is a beautiful language that I love very much. It changed my daily computing life completely. I am thankful to the developers of the JuMP and other related packages. After JuMP, I no longer look for better modeling languages. I am also grateful to Joo Yeon Woo for the cover design and Bomin Kwon for the cat drawing.

Tampa, Florida
Changhyun Kwon

1

Introduction and Installation

This chapter will introduce what the Julia Language is and explain why I love it. More importantly, this chapter will teach you how to obtain Julia and install it in your machine. Well, at this moment, the most challenging task for using Julia in computing would probably be installing the language and other libraries and programs correctly in your own machine. I will go over every step with fine details with screenshots for both Windows and Mac machines. I assumed that Linux users can handle the installation process well enough without much help from this book by reading online manuals and googling. Perhaps the Mac section could be useful to Linux users.

All Julia codes in this book are shared as a git repository and are available at the book website: <http://www.chkwon.net/julia>. Codes are tested with

- `Julia v1.3.0`
- `JuMP v0.21.2`
- `Optim v0.20.6`

I will introduce what JuMP and Optim are gradually later in the book.

1.1. What is Julia and Why Julia?

1.1 What is Julia and Why Julia?

The Julia Language is a young emerging language, whose primary target is technical computing. It is developed for making technical computing more fun and more efficient. There are many good things about the Julia Language from the perspective of computer scientists and software engineers; you can read about the language at [the official website](http://julialang.org)¹.

Here is a quote from the creators of Julia from their first official blog article “[Why We Created Julia](#)”²:

“We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?)”

So this is how Julia was created, to serve all above greedy wishes.

Let me tell you my story. I used to be a Java developer for a few years before I joined a graduate school. My first computer codes for homework assignments and course projects were naturally written in Java; even before then, I used C for my homework assignments for computing when I was an undergraduate student. Later, in the graduate school, I started using MATLAB, mainly because my fellow graduate students in the lab were using MATLAB. I needed to learn from them, so I used MATLAB.

I liked MATLAB. Unlike in Java and C, I don’t need to declare every single variable before I use it; I just use it in MATLAB. Arrays are not just arrays in the computer memory; arrays in MATLAB are just like vectors and matrices. Plotting computation results is easy. For modeling optimization problems, I used GAMS

¹<http://julialang.org>

²<http://julialang.org/blog/2012/02/why-we-created-julia>

and connected with solvers like CPLEX. While the MATLAB-GAMS-CPLEX chain suited my purpose well, I wasn't that happy with the syntax of GAMS—I couldn't fully understand—and the slow speed of the interface between GAMS and MATLAB. While CPLEX provides complete connectivities with C, Java, and Python, it was very basic with MATLAB.

When I finished with my graduate degree, I seriously considered Python. It was—and still is—a very popular choice for many computational scientists. CPLEX also has a better support for Python than MATLAB. Unlike MATLAB, Python is a free and open source language. However, I didn't go with Python and decided to stick with MATLAB. I personally don't like 0 being the first index of arrays in C and Java. In Python, it is also 0. In MATLAB, it is 1. For example, if we have a vector like:

$$\mathbf{v} = \begin{bmatrix} 1 \\ 0 \\ 3 \\ -1 \end{bmatrix}$$

it may be written in MATLAB as:

```
v = [1; 0; 3; -1]
```

The first element of this vector should be accessible by `v(1)`, not `v(0)`. The i -th element must be `v(i)`, not `v(i-1)`. So I stayed with MATLAB.

Later in 2012, the Julia Language was introduced and it looked attractive to me, since at least the array index begins with 1. After some investigations, I still didn't move to Julia at that time. It was ugly in supporting optimization modeling and solvers. I kept using MATLAB.

In 2014, I came across several blog articles and tweets talking about Julia again. I gave it one more look. Then I found a package for modeling optimization problems in Julia, called JuMP—Julia for Mathematical Programming. After spending a few hours, I fell in love with JuMP and decided to go with Julia, well more with JuMP. Here is a part of my code for solving a network optimization problem:

```
@variable(m, 0 <= x[links] <= 1)
@objective(m, Min, sum(c[(i,j)] * x[(i,j)] for (i,j) in links) )
```

```

for i=1:no_node
    @constraint(m, sum(x[(ii,j)] for (ii,j) in links if ii==i )
                - sum(x[(j,ii)] for (j,ii) in links if ii==i ) == b[i])
end

optimize!(m)

```

This is indeed a direct “translation” of the following mathematical language:

$$\min \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij}$$

subject to

$$\begin{aligned} \sum_{(i,j) \in \mathcal{A}} x_{ij} - \sum_{(j,i) \in \mathcal{A}} x_{ji} &= b_i \quad \forall i \in \mathcal{N} \\ 0 \leq x_{ij} \leq 1 &\quad \forall (i,j) \in \mathcal{A} \end{aligned}$$

I think it is a very obvious translation. It is quite beautiful, isn’t it?

CPLEX and its competitor Gurobi are also very smoothly connected with Julia via JuMP. Why should I hesitate? After several years of using Julia, I still love it—I even wrote a book.

1.2 Installing Julia

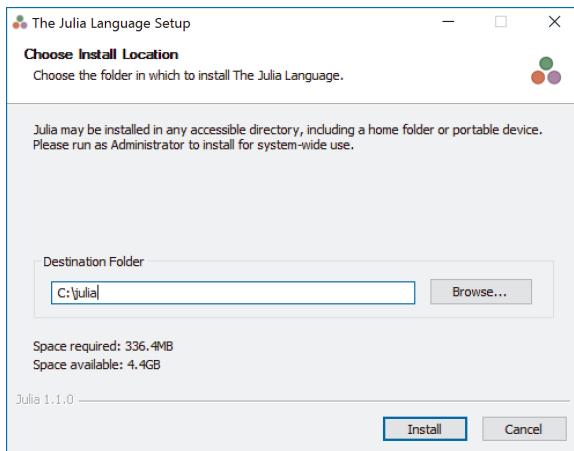
Graduate students and researchers are strongly recommended to install Julia in their local computers. In this guide, we will first install Julia and then install two optimization packages, JuMP and GLPK. JuMP stands for ‘Julia for Mathematical Programming’, which is a modeling language for optimization problems. GLPK is an open-source linear optimization solver that can solve both continuous and discrete linear programs. Windows users go to [Section 1.2.1](#), and Mac users go to [Section 1.2.2](#).

1.2.1 Installing Julia in Windows

- **Step 1.** Download Julia from the official website.³ (Select an appropriate version: 32-bit or 64-bit. 64-bit recommended whenever possible.)

Current stable release (v1.1.0)		
Windows Self-Extracting Archive (.exe) [help]	32-bit	64-bit
Windows 7/Windows Server 2012 users also require Windows Management Framework 3.0 or later		
macOS 10.8+ Package (.dmg) [help]		64-bit
Generic Linux Binaries for x86 [help]	32-bit (GPG)	64-bit (GPG)
Generic FreeBSD Binaries for x86 [help]		64-bit (GPG)
Source	Tarball (GPG)	Tarball with dependencies (GPG) GitHub

- **Step 2.** Install Julia in C:\julia. (You need to make the installation folder consistent with the path you set in Step 3.)

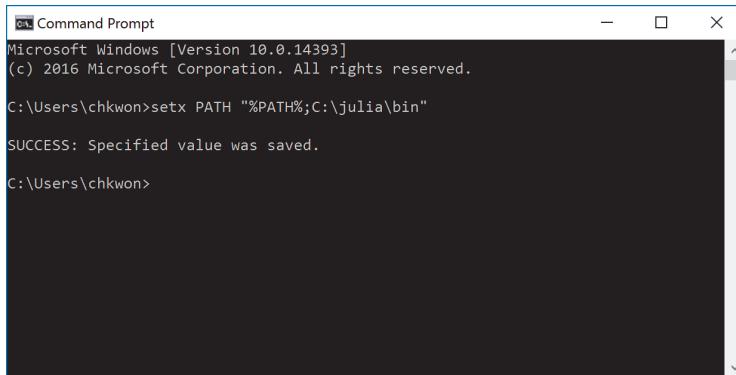


- **Step 3.** Open a Command Prompt and enter the following command:

³<http://julialang.org/downloads/>

1.2. Installing Julia

```
setx PATH "%PATH%;C:\julia\bin"
```

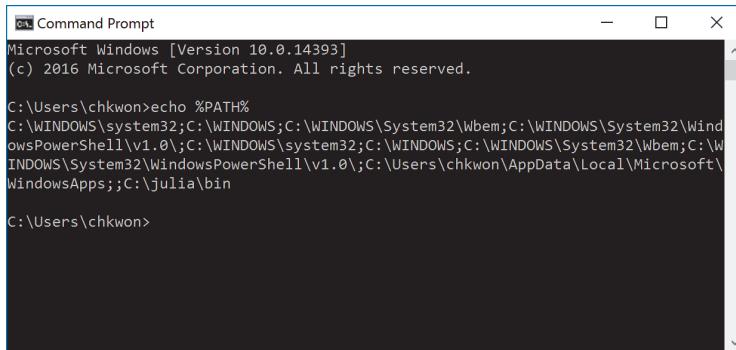


A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following text:
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Users\chkwon>setx PATH "%PATH%;C:\julia\bin"
SUCCESS: Specified value was saved.
C:\Users\chkwon>

If you do not know how to open a Command Prompt, just google ‘how to open command prompt windows.’

- **Step 4.** Open a **NEW** command prompt and type

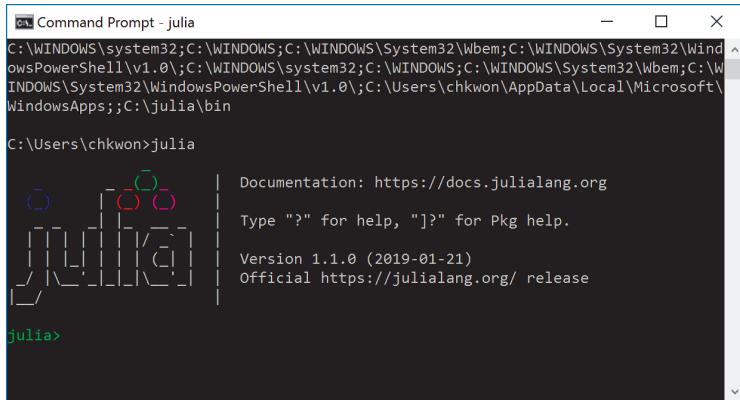
```
echo %PATH%
```



A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following text:
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Users\chkwon>echo %PATH%
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Users\chkwon\AppData\Local\Microsoft\WindowsApps;;C:\julia\bin
C:\Users\chkwon>

The output must include **C:\julia\bin** in the end. If not, you must have something wrong.

- Step 5. Run julia.



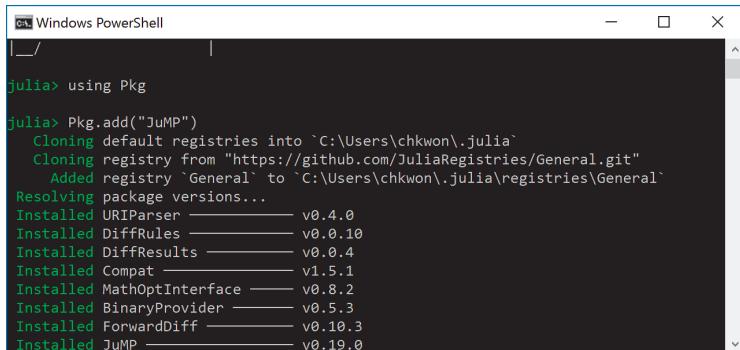
The screenshot shows a Windows Command Prompt window titled "Command Prompt - julia". The path in the title bar is C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Users\chkwon\AppData\Local\Microsoft\WindowsApps;. The command entered is "julia". The output shows the Julia logo, documentation URL (<https://docs.julialang.org>), help instructions ("Type '?' for help, "]?" for Pkg help."), version information ("Version 1.1.0 (2019-01-21)"), and the official release URL (<https://julialang.org/>). The prompt then changes to "julia>".

You have successfully installed the Julia Language on your Windows computer. Now it is time to install additional packages for mathematical optimization.

- Step 6. In your Julia prompt, type

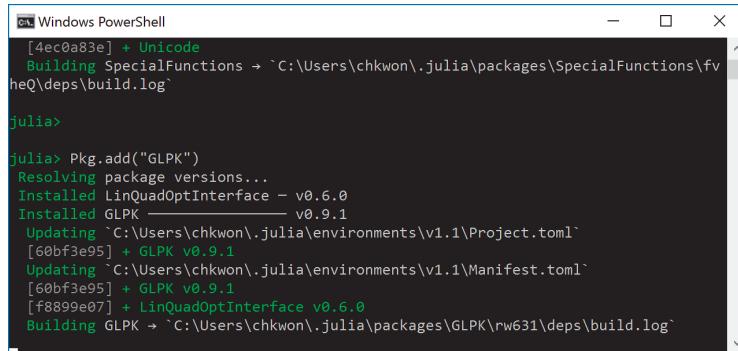
```
julia> using Pkg  
julia> Pkg.add("JuMP")  
julia> Pkg.add("GLPK")
```

Installing the first package can take long time, because it initializes your Julia package folder and synchronizes with the entire package list.



The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command entered is "julia> using Pkg". The output shows the process of adding the "JuMP" package, which involves cloning the default registries into the user's Julia directory and adding the "General" registry. It then resolves package versions and lists the installed packages: URIParser (v0.4.0), DiffRules (v0.0.10), DiffResults (v0.0.4), Compat (v1.5.1), MathOptInterface (v0.8.2), BinaryProvider (v0.5.3), ForwardDiff (v0.10.3), and JuMP (v0.19.0).

1.2. Installing Julia



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command entered is "julia> Pkg.add("GLPK")". The output shows the package being resolved and installed. It lists the installation of "LinQuadOptInterface" v0.6.0, "GLPK" v0.9.1, and "GLPK" v0.9.1 again. It also shows the creation of environments and Manifest.toml files. The final message is "Building GLPK → `C:\Users\chkwon\.julia\packages\GLPK\rw631\deps\build.log`".

- **Step 7.** Open Notepad (or any other text editor such as [Visual Studio Code](#)⁴) and type the following, and save the file as `script.jl` in some folder of your choice.

```
using JuMP, GLPK
m = Model(GLPK.Optimizer)

@variable(m, 0 <= x <= 2 )
@variable(m, 0 <= y <= 30 )

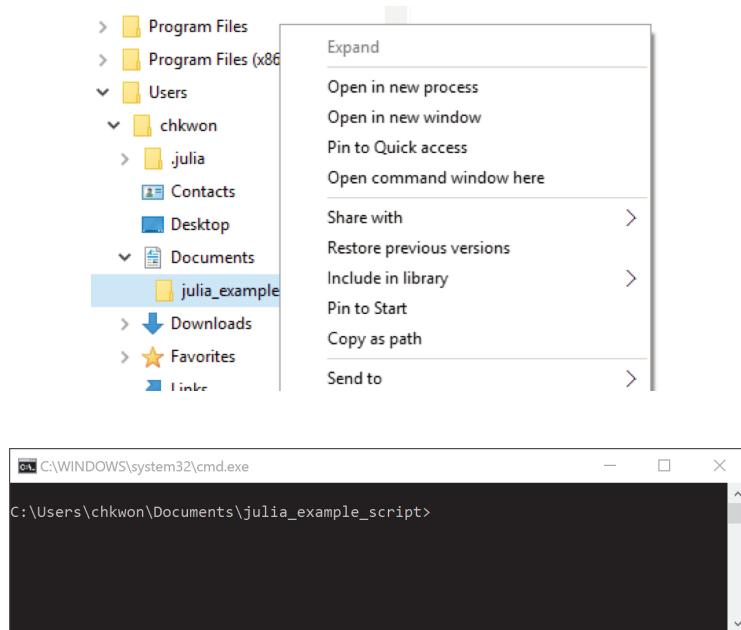
@objective(m, Max, 5x + 3*y )

@constraint(m, 1x + 5y <= 3.0 )

JuMP.optimize!(m)
println("Objective value: ", JuMP.objective_value(m))
println("x = ", JuMP.value(x))
println("y = ", JuMP.value(y))
```

- **Step 8.** Press and hold your **Shift Key** and right-click the folder name, and choose “Open command window here.”

⁴<https://code.visualstudio.com>



- **Step 9.** Type `dir` to see your script file `script.jl`.

```
C:\WINDOWS\system32\cmd.exe
Directory of C:\Users\chkwon\Documents\julia_example_script

02/18/2019  07:40 PM    <DIR>      .
02/18/2019  07:40 PM    <DIR>      ..
02/18/2019  07:40 PM           330 script.jl
                           1 File(s)     330 bytes
                           2 Dir(s)   5,043,027,968 bytes free

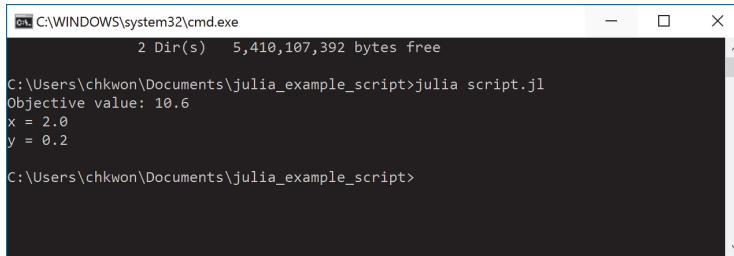
C:\Users\chkwon\Documents\julia_example_script>
```

If you see a filename such as `script.jl.txt`, use the following command to rename:

1.2. Installing Julia

```
ren script.jl.txt script.jl
```

- **Step 10.** Type `julia script.jl` to run your julia script.



A screenshot of a Windows Command Prompt window titled "cmd C:\WINDOWS\system32\cmd.exe". The window shows the following text:
2 Dir(s) 5,410,107,392 bytes free
C:\Users\chkwon\Documents\julia_example_script>julia script.jl
Objective value: 10.6
x = 2.0
y = 0.2
C:\Users\chkwon\Documents\julia_example_script>

After a few seconds, the result of your julia script will be printed. Done.

Please proceed to [Section 1.2.3](#).

1.2.2 Installing Julia in macOS

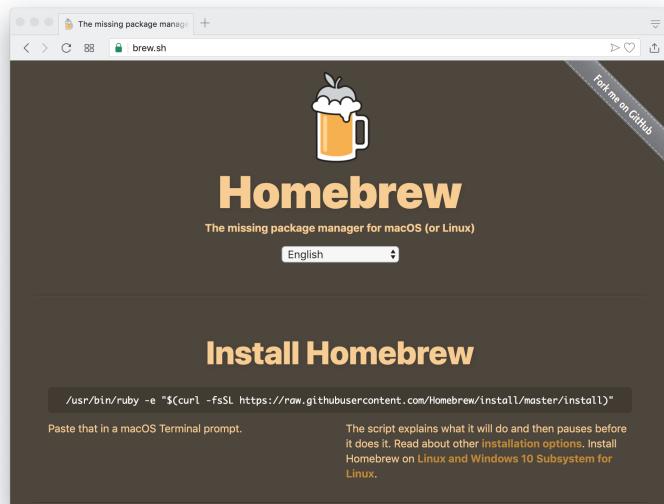
In macOS, we will use a package manager, called [Homebrew](#). It provides a very convenient way of installing software in macOS.

- **Step 1.** Open “Terminal.app” from your Applications folder. (If you do not know how to open it, see [this video](#).⁵ It is convenient to place “Terminal.app” in your dock.



- **Step 2.** Visit <http://brew.sh> and follow the instruction to install Homebrew. It may ask you to enter your password to install Xcode Command Line Tools.

⁵ https://www.youtube.com/watch?v=zw7Nd67_aFw “How to open the terminal window on a Mac”



```
chkwon — bash — 80x24
Last login: Tue Oct 18 11:40:35 on ttys000
chkwon@MacBook:~$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

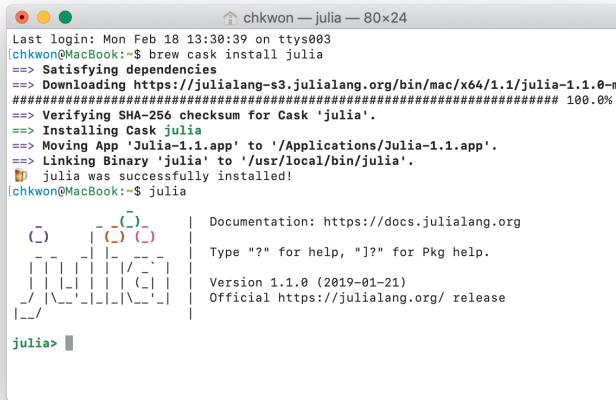
- **Step 3.** Installing Julia using Homebrew: In your terminal, enter the following command:

```
brew cask install julia
```

```
chkwon — bash — 80x7
tests
Already up-to-date.
==> Installation successful!
==> Next steps
Run `brew help` to get started
Further documentation: https://git.io/brew-docs
chkwon@MacBook:~$ brew cask install julia
```

- **Step 5.** In your terminal, enter `julia`.

1.2. Installing Julia



```
Last login: Mon Feb 18 13:30:39 on ttys003
[chkwon@MacBook:~$ brew cask install julia
==> Satisfying dependencies
==> Downloading https://julialang-s3.julialang.org/bin/mac/x64/1.1/julia-1.1.0-m
#####
====SHA-256 checksum for Cask 'julia'.
==> Verifying SHA-256 checksum for Cask 'julia'.
==> Installing Cask julia
==> Moving App 'Julia-1.1.app' to '/Applications/Julia-1.1.app'.
==> Linking Binary 'julia' to '/usr/local/bin/julia'.
[!] julia was successfully installed!
[chkwon@MacBook:~$ julia
julia> ]
```

- **Step 6.** In your Julia prompt, type

```
julia> using Pkg
julia> Pkg.add("JuMP")
julia> Pkg.add("GLPK")
```

Installing the first package can take a long time, because it initializes your Julia package folder and synchronizes with the entire package list.

```
>Last login: Wed Feb 20 22:25:27 on ttys004
[chkwon@MacBook:~]$ julia
Documentation: https://docs.julialang.org
Type "?" for help, "]?" for Pkg help.
Version 1.1.0 (2019-01-21)
Official https://julialang.org/ release

[julia]> using Pkg

[julia]> Pkg.add("JMP")
  Cloning default registries into `~/julia`
  Cloning registry from "https://github.com/JuliaRegistries/General.git"
    Added registry `General` at https://juliaregistries/General
Resolving package versions...
Installed NaNMath └── v0.3.2
Installed DiffResults └── v0.0.4
Installed CommonSubexpressions └── v0.2.0
Installed OrderedCollections └── v1.0.2
Installed BinaryProvider └── v0.5.3
Installed BinDeps └── v0.8.10
```

```
[9a3fb8284] + Random
[ea8e019c] + SHA
[9e88b42a] + Serialization
[1a1011a3] + SharedArrays
[6462feeb] + Sockets
[2f0118ae] + SparseArrays
[10746b16] + Statistics
[8dfed614] + Test
[cfc7118a7] + UUIDs
[4ec0a83e] + Unicode
Building SpecialFunctions → `~/.julia/packages/SpecialFunctions/fvheQdeps/bui
ld.log`
```



```
julia> Pkg.add("GLPK")
Resolving package versions...
Installed LinQuadOptInterface - v0.6.0
Installed GLPK └── v0.9.1
Updating `~/.julia/environments/v1.1/Project.toml`
[660bf3e95] + GLPK v0.9.1
Updating `~/.julia/environments/v1.1/Manifest.toml`
[660bf3e95] + LinQuadOptInterface v0.6.0
Building GLPK → `~/.julia/packages/GLPK/rw631/deps/build.log`
```

- **Step 7.** OpenTextEdit (or any other text editor such as [Visual Studio Code⁶](#)) and type the following, and save the file as `script.jl` in some folder of your choice.

⁶<https://code.visualstudio.com>

1.2. Installing Julia

```
using JuMP, GLPK
m = Model(GLPK.Optimizer)

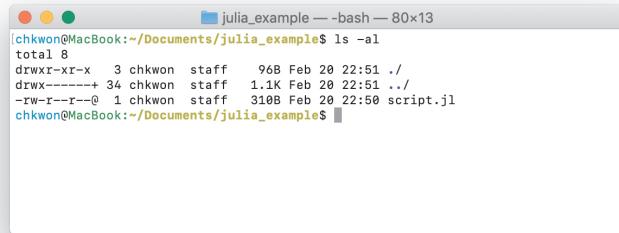
@variable(m, 0 <= x <= 2 )
@variable(m, 0 <= y <= 30 )

@objective(m, Max, 5x + 3*y )

@constraint(m, 1x + 5y <= 3.0 )

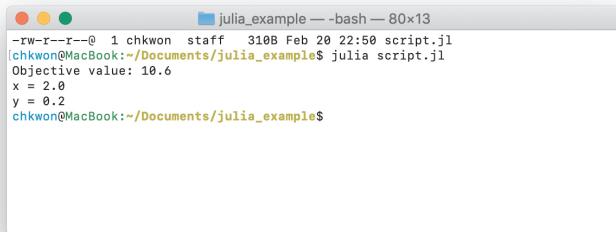
JuMP.optimize!(m)
println("Objective value: ", JuMP.objective_value(m))
println("x = ", JuMP.value(x))
println("y = ", JuMP.value(y))
```

- **Step 8.** Open a terminal window⁷ at the folder that contains your `script.jl`.
- **Step 9.** Type `ls -al` to check your script file.



- **Step 10.** Type `julia script.jl` to run your script.

⁷To do this, you can drag the folder to the Terminal.app icon in your dock, or see <http://osxdaily.com/2011/12/07/open-a-selected-finder-folder-in-a-new-terminal-window/>



```
julia_example -- bash -- 80x13
-rw-r--r--@ 1 chkwon  staff  3108 Feb 20 22:50 script.jl
[chkwon@MacBook:~/Documents/julia_example$ julia script.jl
Objective value: 10.6
x = 2.0
y = 0.2
chkwon@MacBook:~/Documents/julia_example$ ]
```

After a few seconds, the result of your Julia script will be printed. Done.

Please proceed to [Section 1.2.3](#).

1.2.3 Running Julia Scripts

When you are ready, there are basically two methods to run your Julia script:

- In your Command Prompt or Terminal, enter `C:> julia your-script.jl`
- In your Julia prompt, enter `julia> include("your-script.jl")`.

1.2.4 Installing Gurobi

Instead of GLPK, one can use Gurobi, which is a commercial optimization solver package for solving LP, MILP, QP, MIQP, etc. Gurobi is free for students, teachers, professors, or anyone else related to educational organizations.

To install, follow these steps:

1. [Download Gurobi Optimizer](#)⁸ and install in your computer. (You will need to register as an academic user.)
2. [Request a free academic license](#)⁹ and follow their instructions to activate it.

⁸<https://www.gurobi.com/downloads/gurobi-optimizer-eula/>

⁹<https://www.gurobi.com/academia/academic-program-and-licenses/>

1.2. Installing Julia

3. Run Julia and add the Gurobi package. You need to tell Julia where Gurobi is installed:

On Windows:

```
julia> ENV["GUROBI_HOME"] =
           "C:\\Program Files\\gurobi910\\win64"
julia> using Pkg
julia> Pkg.add("Gurobi")
```

On macOS:

```
julia> ENV["GUROBI_HOME"] =
           "/Library/gurobi910/mac64"
julia> using Pkg
julia> Pkg.add("Gurobi")
```

4. Ready. Test the following code:

```
using JuMP, Gurobi
m = Model(Gurobi.Optimizer)
@variable(m, x <= 5)
@variable(m, y <= 45)
@objective(m, Max, x + y)
@constraint(m, 50x + 24y <= 2400)
@constraint(m, 30x + 33y <= 2100)

JuMP.optimize!(m)
println("Objective value: ", JuMP.objective_value(m))
println("x = ", JuMP.value(x))
println("y = ", JuMP.value(y))
```

1.2.5 Installing CPLEX

Instead of Gurobi, you can install and connect the CPLEX solver, which is also free to academics.

You can follow this step by step guide to install:

1. Go to the IBM ILOG CPLEX Optimization Studio page¹⁰.
2. Click ‘Access free academic edition.’
3. Log in with your institution email and certify.
4. Download an appropriate version of IBM ILOG CPLEX Optimization Studio.
It should be v12.10 or higher.
5. Run the downloaded file and install CPLEX. I recommend using the default installation folder.
6. Add the CPLEX package in Julia. You have to tell Julia where the CPLEX library is installed.

On Windows:

```
julia> ENV["CPLEX_STUDIO_BINARIES"] =  
"C:\\Program Files\\CPLEX_Studio1210\\cplex\\bin\\x86-64_win\\"  
julia> using Pkg  
julia> Pkg.add("CPLEX")  
julia> Pkg.build("CPLEX")
```

On macOS:

```
julia> ENV["CPLEX_STUDIO_BINARIES"] =  
"/Applications/CPLEX_Studio1210/cplex/bin/x86-64_osx/"  
julia> using Pkg  
julia> Pkg.add("CPLEX")  
julia> Pkg.build("CPLEX")
```

7. Ready. Test the following code:

```
using JuMP, CPLEX  
m = Model(CPLEX.Optimizer)  
@variable(m, x <= 5)  
@variable(m, y <= 45)
```

¹⁰<https://www.ibm.com/products/ilog-cplex-optimization-studio>

1.3. Installing IJulia

```
@objective(m, Max, x + y)
@constraint(m, 50x + 24y <= 2400)
@constraint(m, 30x + 33y <= 2100)

JuMP.optimize!(m)
println("Objective value: ", JuMP.objective_value(m))
println("x = ", JuMP.value(x))
println("y = ", JuMP.value(y))
```

1.3 Installing IJulia

You can also use an interactive Julia environment in your local computer, called [Jupyter Notebook](#)¹¹. Well, at first there was [IPython](#) notebook that was an interactive programming environment for the Python language. It has been popular, and now it is extended to cover many other languages such as R, Julia, Ruby, etc. The extension became the Jupyter Notebook project. For Julia, it is called [IJulia](#), following the naming convention of [IPython](#).

To use [IJulia](#), we need a distribution of Python and Jupyter. Julia can automatically install a distribution for you, unless you want to install it by yourself. If you let Julia install Python and Jupyter, they will be private to Julia, i.e. you will not be able to use Python and Jupyter outside of Julia.

The following process will automatically install Python and Jupyter.

1. Open a new terminal window and run Julia. Initialize environment variables:

```
julia> ENV["PYTHON"] = ""
"""

julia> ENV["JUPYTER"] = ""
"""
```

2. Install [IJulia](#):

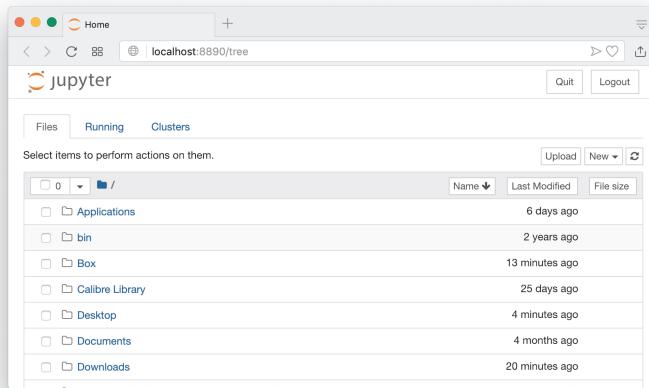
¹¹<http://jupyter.org>

```
julia> using Pkg  
julia> Pkg.add("IJulia")
```

3. To open the IJulia notebook in your web browser:

```
julia> using IJulia  
julia> notebook()
```

It will open a webpage in your browser that looks like the following screenshot:



The current folder will be your home folder. You can move to another folder and also create a new folder by clicking the “New” button on the top-right corner of the screen. After locating a folder you want, you can now create a new IJulia notebook by clicking the “New” button again and select the julia version of yours, for example “Julia 1.1.0”. See [Figure 1.1](#).

It will basically open an interactive session of the Julia Language. If you have used Mathematica or Maple, the interface will look familiar. You can test basic Julia commands. When you need to evaluate a block of codes, press **Shift+Enter**, or press the “play” button. See [Figure 1.2](#).

If you properly install a plotting package like PyPlot (details in [Section 3.10.1](#)), you can also do plotting directly within the IJulia notebook as shown in [Figure 1.4](#).

1.3. Installing IJulia

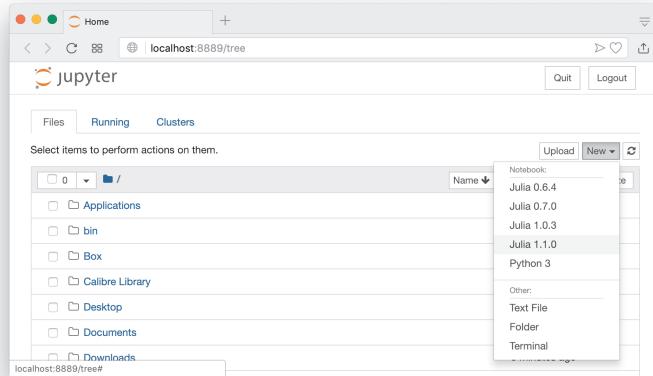


Figure 1.1: Creating a new notebook

A screenshot of a Jupyter Notebook window titled 'Untitled2'. The URL is 'localhost:8889/notebooks/Untitled2.ipynb'. The notebook contains three code cells:

```
In [1]: 1+1
Out[1]: 2
```

```
In [2]: println("Hello IJulia!")
Hello IJulia!
```

```
In [3]: f(x,y) = x^2 + y^2
f(2,3)
Out[3]: 13
```

The status bar at the bottom right indicates 'Julia 1.1.0'.

Figure 1.2: Some basic Julia codes.



Figure 1.3: This is the REPL.

Personally, I prefer the REPL for most tasks, but I do occasionally use IJulia, especially when I need to test some simple things and need to plot the result quickly, or when I need to share the result of Julia computation with someone else. (IJulia can export the notebook in various formats, including HTML and PDF.)

What is REPL? It stands for read-eval-print loop. It is the Julia session that runs in your terminal; see Figure 1.3, which must look familiar to you already.

1.4 Package Management

There are many useful packages in Julia and we rely many parts of our computations on packages. If you have followed my instructions to install Julia, JuMP, Gurobi, and CPLEX, you have already installed a few packages. There are some more commands that are useful in managing packages.

```
julia> using Pkg
julia> Pkg.add("PackageName")
```

1.4. Package Management

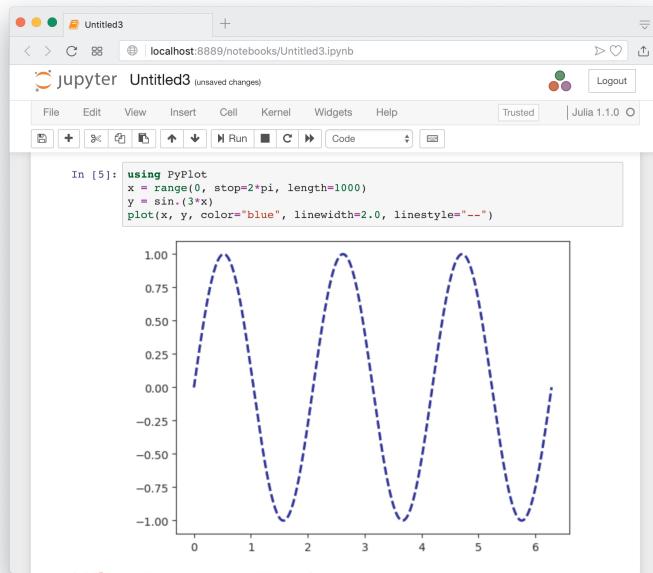


Figure 1.4: Plotting in IJulia

```
Last login: Wed Feb 20 13:39:59 on ttys001
[chkwon@MacBook:~$ julia
Documentation: https://docs.julialang.org
Type "?" for help, "]?" for Pkg help.

Version 1.1.0 (2019-01-21)
Official https://julialang.org/ release

[julia> 1+1
2

(v1.1) pkg>
```

Figure 1.5: Package Mode in REPL

This installs a package, named `PackageName`. To find its online repository, you can just google the name `PackageName.jl`, and you will be directed to a repository hosted at GitHub.com.

Using `Pkg.add` requires `using Pkg` first. In REPL, by pressing the ‘]’ key, you can enter the package management mode (Figure 1.5) and the prompt will change as follows:

```
(v1.3) pkg>
```

Then to install a package you can simply enter:

```
(v1.3) pkg> add PackageName
```

To install the JuMP package, you can do:

1.4. Package Management

```
(v1.3) pkg> add JuMP
```

To come back to the julia prompt, press the backspace or delete key.

```
julia> Pkg.rm("PackageName")
(v1.3) pkg> rm PackageName
```

This removes the package.

```
julia> Pkg.update()
(v1.3) pkg> update
```

This updates all packages that are already installed in your machine to the most recent versions.

```
julia> Pkg.status()
(v1.3) pkg> status
```

This displays what packages are installed and what their versions are. If you just want to know the version of a specific package, you can do:

```
julia> Pkg.installed()["PackageName"]
```

```
julia> Pkg.build("PackageName")
(v1.3) pkg> build PackageName
```

Occasionally, installing a package will fail during the `Pkg.add("PackageName")` process, usually because some libraries are not installed or system path variables are not configured correctly. Try to install some required libraries again and check the system path variables first. Then you may need to reboot your system or restart your Julia session. Then `Pkg.build("PackageName")`. Since you have downloaded package files during `Pkg.build("PackageName")`, you don't need to download them again; you just build it again.

1.5 Help

In REPL, you can use the Help mode. By pressing the ? key in REPL, you can enter the help mode. The prompt will change as follows:

```
help?>
```

Then type in any function name, for example, `println`, which results in:

```
help?> println
search: println printstyled print sprint isprint

println([io::IO], xs...)

Print (using print) xs followed by a newline. If io is not supplied, prints to
stdout.

Examples

julia> println("Hello, world")
Hello, world

julia> io = IOBuffer();

julia> println(io, "Hello, world")

julia> String(take!(io))
"Hello, world\n"
```

See also [Figure 1.6](#).

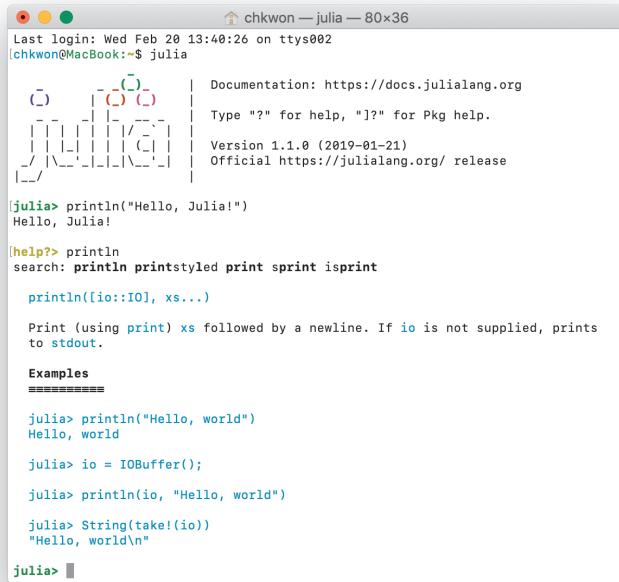
Readers can find codes and other helpful resources in the author's website at

<http://www.chkwon.net/julia>

which also includes a link to a Facebook page of this book for discussion and communication.

This book does *not* teach everything of the Julia Language—only a very small part of it. When you want to learn more about the language, the first place you need to visit is

1.5. Help



The screenshot shows a terminal window titled "chkwon — julia — 80x36". The window displays the Julia REPL and its help documentation. At the top, it shows the last login information: "Last login: Wed Feb 20 13:40:26 on ttys002" and the command "chkwon@MacBook:~\$ julia". Below this is the help documentation for the `print` family of functions, which includes the following text:

```
Documentation: https://docs.julialang.org
Type "?" for help, "]?" for Pkg help.
Version 1.1.0 (2019-01-21)
Official https://julialang.org/ release

(julia> println("Hello, Julia!")
Hello, Julia!

[help> println
search: println printstyled print sprint isprint

println([io::IO], xs...)
Print (using print) xs followed by a newline. If io is not supplied, prints to stdout.

Examples
=====
julia> println("Hello, world")
Hello, world

julia> io = IOBuffer();
julia> println(io, "Hello, world")
julia> String(take!(io))
"Hello, world\n"

julia>
```

Figure 1.6: Help Mode in REPL

<http://julialang.org/learning/>

where many helpful books, tutorials, videos, and articles are listed. Also, you will need to visit the official documentation of the Julia Language at

<http://docs.julialang.org/>

which I think serves as a good tutorial as well.

When you have a question, there will be many Julia enthusiasts ready for you. For questions and discussion, visit

<https://discourse.julialang.org>

and

<http://julialang.org/community/>

You can also ask questions at <http://stackoverflow.com> with tag `julia-lang`.

The webpage of JuMP is worth visiting for information about the JuMP.jl package.

<http://jump.dev>

1.5. Help
