

# Julia/IJulia/JuMP Installation Guide

This document was borrowed and modified from <http://www.juliaopt.org/install.pdf>  
by Laurent Lessard (Jan 16, 2017)

## Purpose

This document aims to get you set up with




1. **Julia**: the programming language.
2. **IJulia**: a popular environment for writing Julia code in.
3. **JuMP**: a package for Julia that lets you express optimization models, *and* **CoinOR Clp and Cbc**: open-source solvers for LPs and MILPs, *and* **Ipopt**: open-source solver for nonlinear programming.
4. **Learning Julia**.

**Julia** (<http://julialang.org/>) is a relatively new programming language that is aimed at “technical computing” the kind of computing that most of you do every day. It shares many similarities with both Matlab and numerical Python. We’ll be using Julia through the **IJulia** notebook interface, which lets you mix code, documentation, mathematics, graphics, etc.

**JuMP** (<https://github.com/JuliaOpt/JuMP.jl>) is a modeling language for optimization problems. It lets you translate the mathematical statement of an optimization problem into a form the computer can work with, but is still easy for humans to work with. It is developed by students at the MIT Operations Research Center, so we can provide support for you if you want to use it in your work. If you want to get involved in its future development, please get in touch!

## 1. Julia

You can download Julia from the official Julia downloads page: <http://julialang.org/downloads/>. You should download the “**current release**” (**v0.5.0**). (If you have a version of Julia already installed which is older than v0.5.0, please update it.) Accept the default setup options.

-  If you are on Windows you should probably choose the 64bit version. If you have a very old Windows computer (>5 years old) there is a chance you will have a 32bit version of Windows. To check, go to the “System Information” page (start typing “system information” in the Start menu and it should show up).
-  If you have an old Mac and haven’t upgraded to OS X 10.7 or higher, you won’t be able to run Julia. You can check this by clicking the Apple logo in the top-left of your screen and clicking “About This Mac”
-  Uptodate versions of Julia are available for some distributions. Please read instructions on the download page.

## 2. IJulia

IJulia is a browser-based environment for writing and running Julia code. It lets you combine code, text, graphics, and equations all in one place. You may have heard of IPython, which is something similar for the Python language. Both IJulia and IPython use the same infrastructure.

- Open Julia by double-clicking the icon or from the Start menu (for Windows) or in your Applications folder (Apple).
- A window with a `julia>` prompt will appear.
- Type `Pkg.add("IJulia")` and wait for the installation to finish.

### After installing IJulia

The easiest way to check if your installation was successful is to open Julia and run

```
using IJulia
notebook()
```

If you encounter problems, check the troubleshooting section of the IJulia website:

<https://github.com/JuliaLang/IJulia.jl>. You can use `notebook(detached=true)` to launch a notebook server in the background that will persist even when you quit Julia. In any case, a web browser window should open, with something that looks like:



(the list of files will be different). Click “New” and select “Julia 0.5.0”. A new tab will open. The new window should contain a box: “In [ ]”:

```
In [ ]:
```

type `1+1` and press `shift + enter/return` at the same time. The result should appear like

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

In [ ]:
```

you can also run IJulia from a command prompt (either Windows or Mac OS) by executing the command `jupyter notebook`.

### 3. JuMP, Coin-OR Clp/Cbc, Ipopt

- Open an IJulia notebook as described above.
- In the first cell enter the following commands:

```
Pkg.add("JuMP")
Pkg.add("Clp")
Pkg.add("Ipopt")
```

Then press shift + enter/return to run the cell. NOTE: these commands are case-sensitive, so it's important that you type "JuMP" and not "jump" or "JUMP".

- This will install the JuMP package, the Clp package and solver, the Cbc package (which is a dependency of the Clp package), and Ipopt solver. It might take a while as it has to download the code and solver (on Linux it will build them from source).
- Test it worked by writing in the next cell:

```
using JuMP
m = Model()
@variable(m, x[1:2], Bin)
@constraint(m, x[1] + x[2] <= 1)
@objective(m, Max, 3x[1] + 2x[2])
solve(m)
print(getvalue(x))
```

and executing the cell as before..

- You should see something like

```
In [3]: using JuMP
m = Model()
@variable(m, x[1:2], Bin)
@constraint(m, x[1] + x[2] <= 1)
@objective(m, Max, 3x[1] + 2x[2])
solve(m)
print(getvalue(x))

[1.0,0.0]
```

### 4. Now familiarize yourself with Julia

- Class time is limited, so the focus will mostly be on modeling optimization problems with JuMP. We'll assume you are familiar with the basics of Julia to save time. We will have a dedicated Julia tutorial in the second week of class but otherwise you'll be expected to know your way around Julia and learn on your own as the class progresses
- The good news is that if you have used a language like Matlab or Python before then you pretty much know the Julia basics already!
- There are lots of resources out there to learn Julia:
  - Julia video tutorial: <https://www.youtube.com/watch?v=vWkgEddb4A>
  - Julia + IJulia cheatsheet: <http://math.mit.edu/~stevenj/Juliacheatsheet.pdf>
  - Learn X in Y minutes, Julia edition: <http://learnxinyminutes.com/docs/julia/>
  - And of course, the detailed Julia manual: <http://docs.julialang.org/en/latest/>