Introduction to the Julia Language

- About Julia
- How to get it
- Basic operations and documentation
- Functions

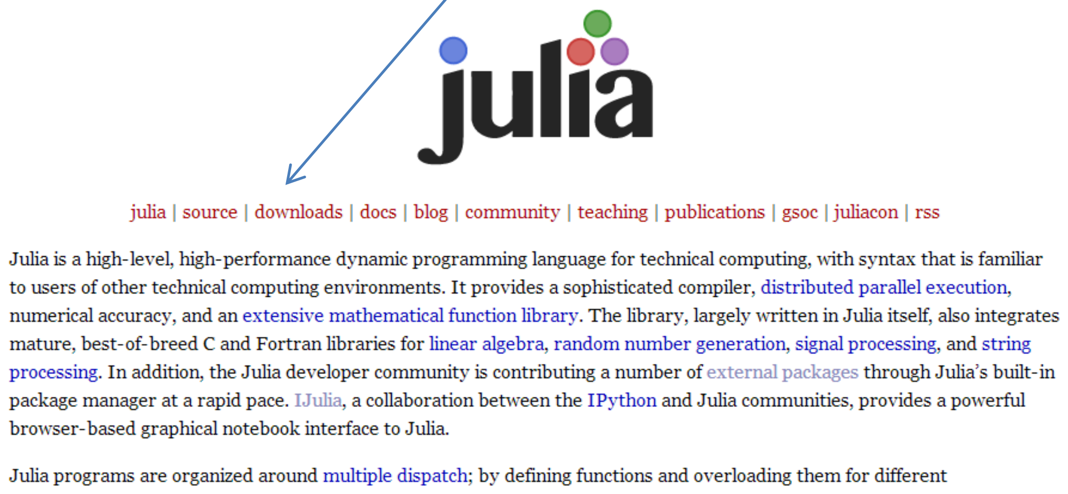
About Julia

- Released in 2012
- Geared towards numerical (math) and scientific computing
- Open-source, meaning it’s free and anyone can contribute
- A solid, flexible, powerful language, easy to learn and use but also high-level and fast.

How to Access Julia

The very basics:

Go to julialang.org, click on downloads, and then download the version suitable for your machine.



julia

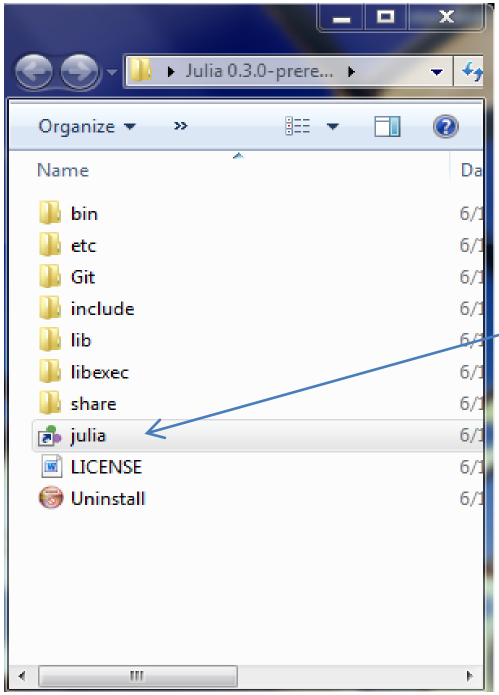
julia | source | downloads | docs | blog | community | teaching | publications | gsoc | juliacon | rss

Julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing environments. It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. The library, largely written in Julia itself, also integrates mature, best-of-breed C and Fortran libraries for linear algebra, random number generation, signal processing, and string processing. In addition, the Julia developer community is contributing a number of external packages through Julia's built-in package manager at a rapid pace. IJulia, a collaboration between the IPython and Julia communities, provides a powerful browser-based graphical notebook interface to Julia.

Julia programs are organized around multiple dispatch; by defining functions and overloading them for different

How to Access Julia

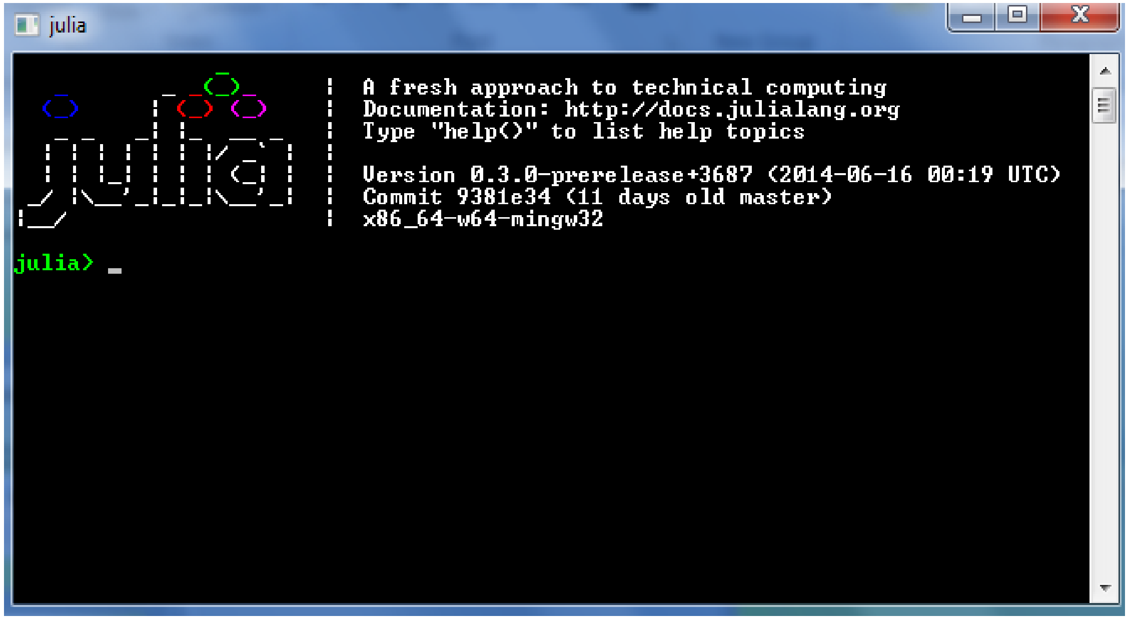
After you install it, you will have a folder somewhere that opens to look like this:



bin 6/1
etc 6/1
Git 6/1
include 6/1
lib 6/1
libexec 6/1
share 6/1
julia 6/1
LICENSE 6/1
Uninstall 6/1

Double-click on the “julia” file

How to Access Julia



```
               _
   _       _ _(_)_     |  A fresh approach to technical computing
  (_)     | (_) (_)    |  Documentation: http://docs.julialang.org
   _ _   _| |_  __ _   |  Type "help()" to list help topics
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 0.3.0-prerelease+3687 (2014-06-16 00:19 UTC)
 _/ |\__'_|_|_|\__'_|  |  Commit 9381e34 (11 days old master)
|__/                   |  x86_64-w64-mingw32

julia> _
```

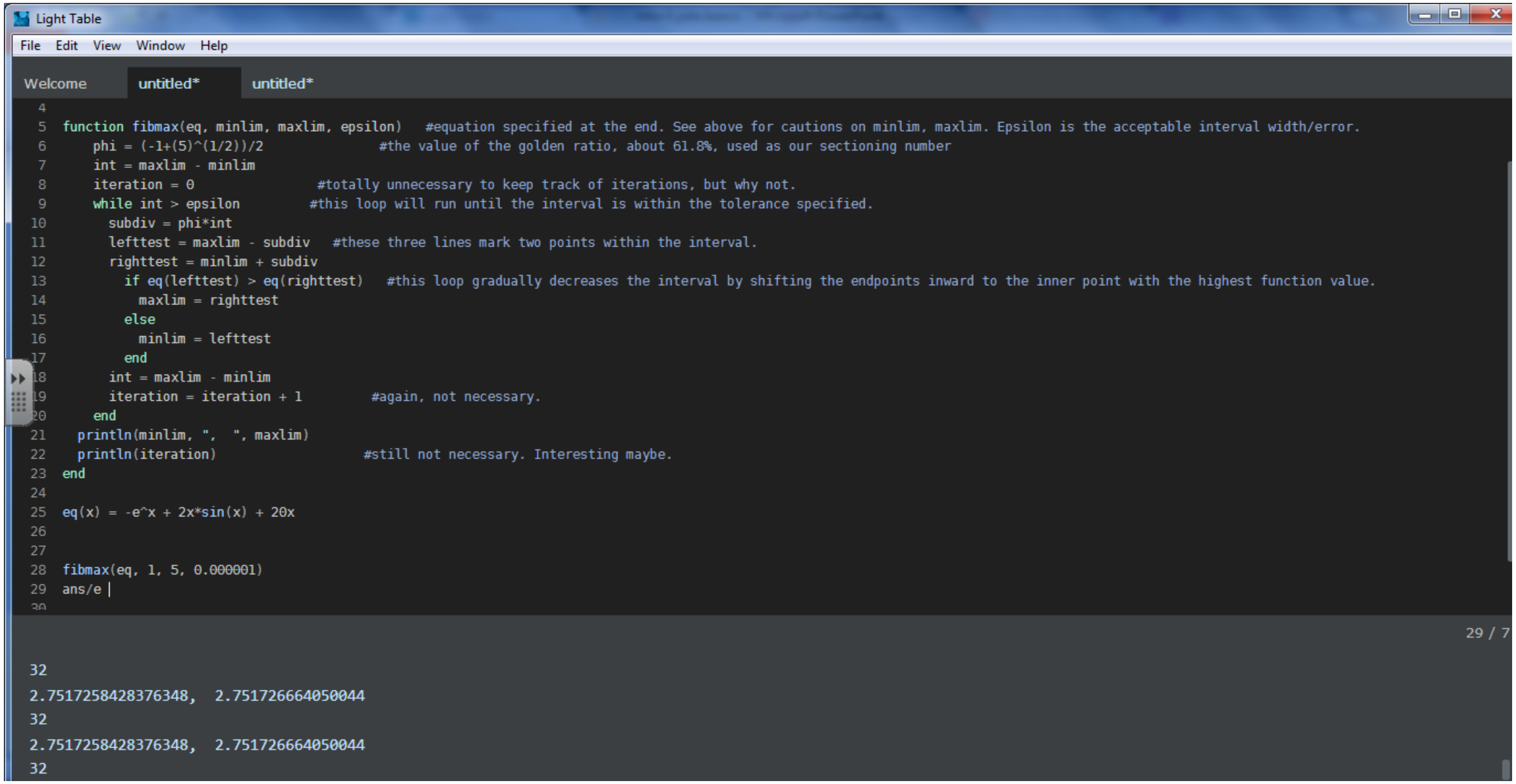
This is what you’ll see. Type in 1 + 1 and hit “enter”. Play for a bit.

How to Access Julia

The command-prompt interface is very bare-bones. There are some free IDEs (integrated development environment)s that are a little “nicer”, for example allowing mouse usage, multiple tabs and easier saving. Check out:

- lighttable.com
- forio.com/products/julia-studio/download

IDE from www.lighttable.com

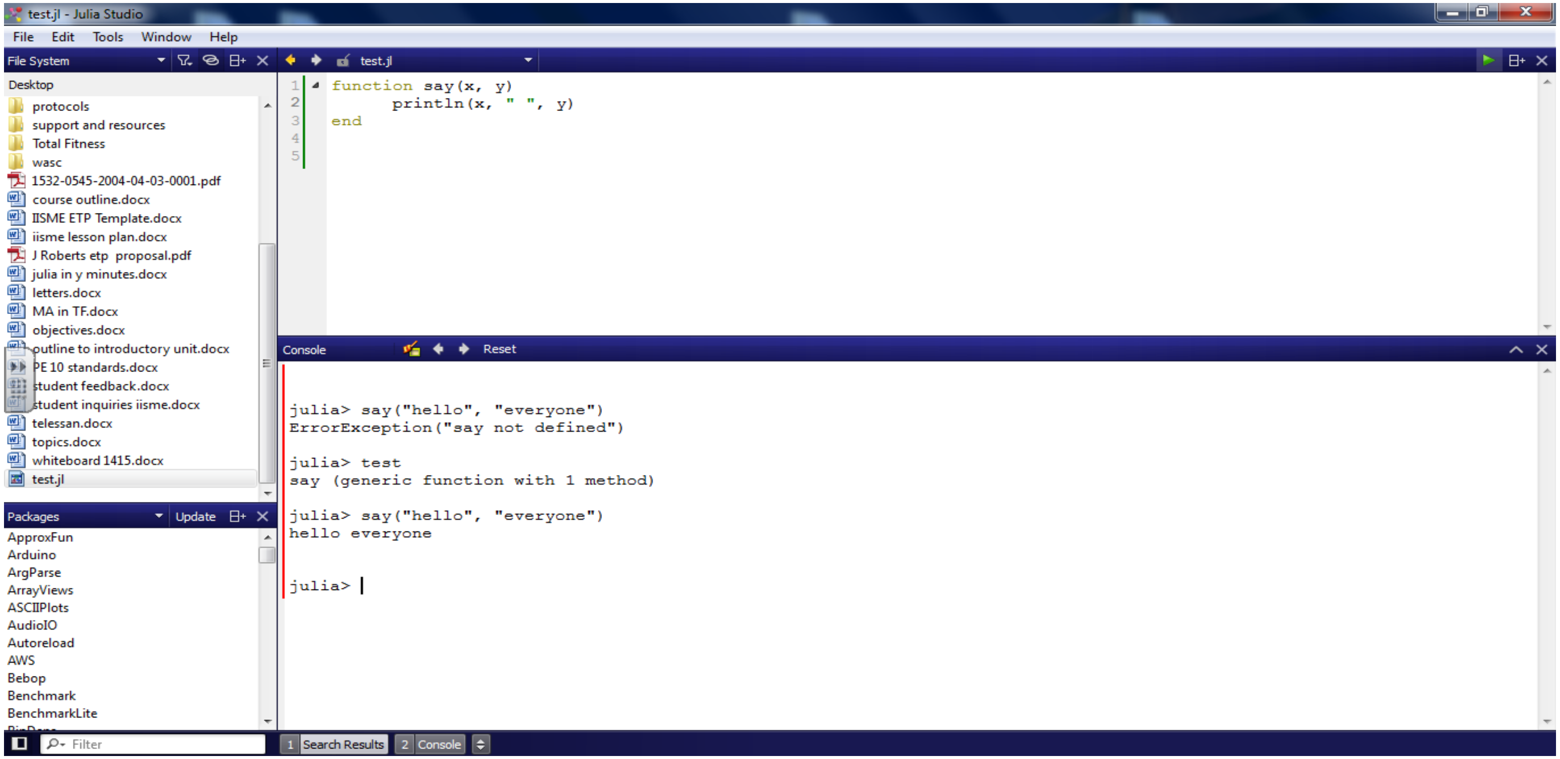


```

function fibmax(eq, minlim, maxlim, epsilon)   #equation specified at the end. See above for cautions on minlim, maxlim. Epsilon is the acceptable interval width/error.
    phi = (-1+(5)^(1/2))/2               #the value of the golden ratio, about 61.8%, used as our sectioning number
    int = maxlim - minlim
    iteration = 0                #totally unnecessary to keep track of iterations, but why not.
    while int > epsilon         #this loop will run until the interval is within the tolerance specified.
      subdiv = phi*int
      lefttest = maxlim - subdiv   #these three lines mark two points within the interval.
      righttest = minlim + subdiv
        if eq(lefttest) > eq(righttest)   #this loop gradually decreases the interval by shifting the endpoints inward to the inner point with the highest function value.
          maxlim = righttest
        else
          minlim = lefttest
        end
      int = maxlim - minlim
      iteration = iteration + 1         #again, not necessary.
    end
  println(minlim, ",  ", maxlim)
  println(iteration)                   #still not necessary. Interesting maybe.
end

eq(x) = -e^x + 2x*sin(x) + 20x


fibmax(eq, 1, 5, 0.000001)
ans/e
```

```
32
2.7517258428376348,  2.751726664050044
32
2.7517258428376348,  2.751726664050044
32
```

Lighttable is a better interface, but requires some behind-the-scenes setup.

Julia Studio IDE from forio.com



```julia
function say(x, y)
      println(x, " ", y)
end
```

```
julia> say("hello", "everyone")
ErrorException("say not defined")

julia> test
say (generic function with 1 method)

julia> say("hello", "everyone")
hello everyone


julia> 
```

Julia Studio is a little clunkier, but requires very little setup.

Some Math Operations in Julia

Operation	Looks like	response
Adding	-3 + 4	1
Subtracting	2 - 4	-2
Multiplying	2 * 3	6
Dividing	5/3	1.666666667
Rounding	round(5/3,2)	1.67
Exponent	2^5	32
Square root	sqrt(25)	5.0
Make a fraction	4//3	4//3
Find remainder	126%8	6

Longer Math Operations

Try these:

```
3*4 + 6^2 - 12 + sin(2pi/3)    36.866
(5 + 3im)*(6 - 2im)             36 + 8im
f(x) = x^2 + 8x - 12 f(generic func...)
f(2)                            8
f(-1)                           -19
f(e)                        17.1353...
rad2deg(pi/2)                   90.0
deg2rad(60)                     1.04…
```

Variables

Variables can be named almost anything.

```
x = 2                      2
  3x + 7                   13
puppies = 6                6
  x + puppies              8
ans + 2                    10
y = ans                    10
  y                        10
puppies = x                2
  puppies                  2
  x                        2
```

One-line functions

```
g(x, y) = x + y          g(generic function…)
   g(6, 12)                   18
f(x) = 2x + 7            f(generic function…)
   x = 8                      8
   f(x)                       23
   g(x, 2)                    10
x = f(x)                      23
x                             23
x = x + 4                     27
x                             27
x + 4 = x                     ERROR
```

Practice Problem 1

1. Suppose you want to find the slope between two points (a, f(a)) and (b, f(b)) where $f(x) = x^2 + 2x - 7$.

a) Enter the one-line function $f(x) = x^2 + 2x - 7$.

b) Create a function g(a, b) that will find the slope between two points x = a and x = b on f(x).

c) Use g to find the slope between x = 3 and x = 5

d) Use g to find the slope between x = -2 and x = 7

Output

The command “println” will cause Julia to give output. Try these:

```
x = 2                          2
println(x)                     2
println("x")                   x
println("hello")               hello
f(x) = x^2             f(generic func…)
println(“8 squared is $(f(8))”)
                  8 squared is 64
println(“$x squared is $(f(x))”)
                      2 squared is 4
```

Multi-line Functions

Here’s another way to write the function for practice problem 1:

```
f(x) = x^2 + 2x - 7           f(generic…)
function slope(f, a, b)
     y1 = f(a)
     y2 = f(b)
     m = (f(a) - f(b))/(a-b)
     println("the slope is $m”)
end                     slope(generic…)
slope(f, 4, 6)          the slope is 12.0
```

Navigating in Julia

By now, you’ve probably screwed up once and gotten the dreaded `ERROR` message (in red caps, no less!) You don’t need to retype.

Use the up-arrow key to recall the last line you typed. You can navigate with arrows.

On a multi-line function, hitting “enter” will submit it again unless you delete the “end” command first while editing.

Navigating in Julia

Type this in:

```
function oops(x, y)
    w = x + a
    p = y - 2
    println(w + p)
end                    oops(generic…)

oops(3, 4)   ERROR: a not defined…
```

Navigating in Julia

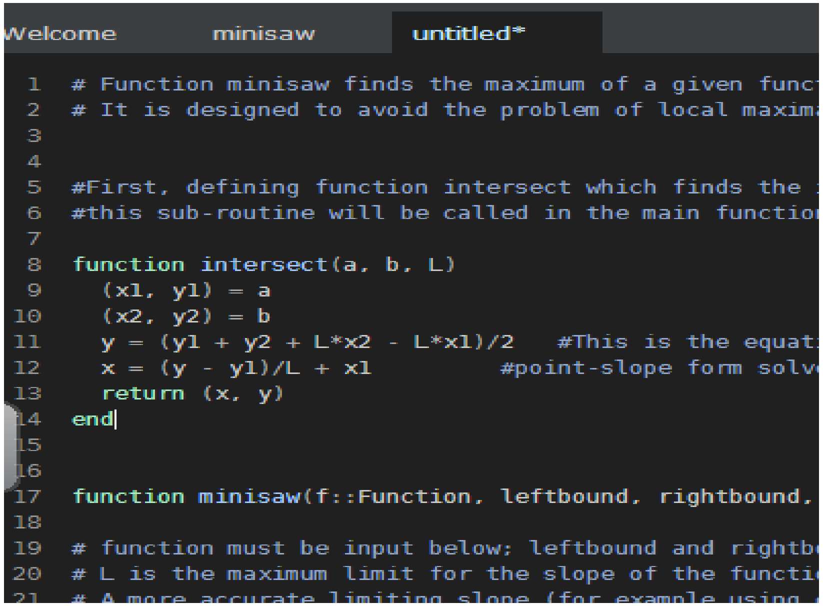
To fix it,

1. Use your up arrow until you recall that section of code.
2. Arrow down and delete the “`end`” at the end.
3. Arrow back up. After the first line hit enter, type in “`a = 2`”, then replace the “`end`”.

Now try `oops(3, 5).` You should get `8`.

Documenting in Julia

Most languages have a system by which you can type commentary into your code without it being compiled along with the code. In Julia, the symbol is a hashtag, #. Anything after a hashtag on a single line will be ignored by the program.



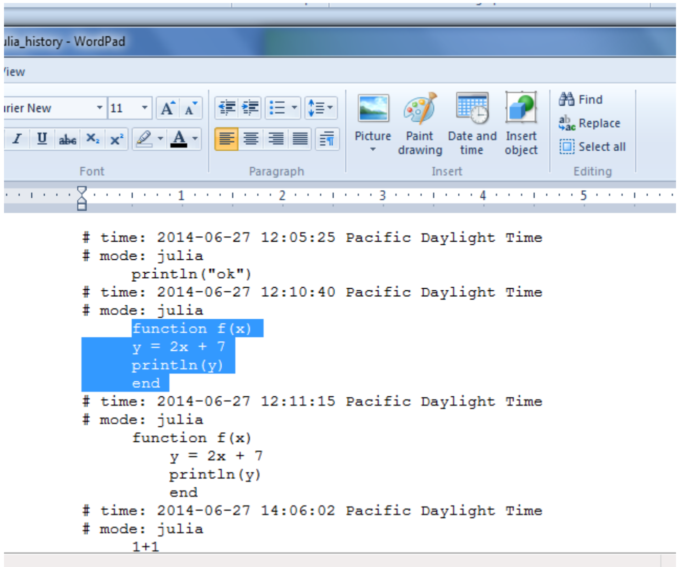
```
# Function minisaw finds the maximum of a given func
# It is designed to avoid the problem of local maxima


#First, defining function intersect which finds the
#this sub-routine will be called in the main functio

function intersect(a, b, L)
  (x1, y1) = a
  (x2, y2) = b
  y = (y1 + y2 + L*x2 - L*x1)/2   #This is the equat
  x = (y - y1)/L + x1         #point-slope form solv
  return (x, y)
end


function minisaw(f::Function, leftbound, rightbound,

# function must be input below; leftbound and rightb
# L is the maximum limit for the slope of the functi
# A more accurate limiting slope (for example using
```

With small, simple functions there's no real reason to document your code, but longer functions may involve instructions that you might forget or other people might want to know.

Saving in Julia

If you’re using an IDE, saving will be taken care of for you.

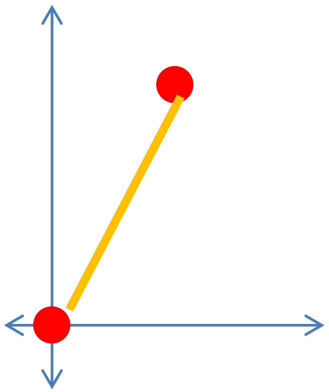
If you’re using basic Julia, it’s also taken care of, but harder to find. If you search for julia_history you will find a document (you can open it in your notepad, but you’ll have to close the julia program) with everything you’ve ever entered. It’s all there. You can delete, modify, copy, save, paste, print, whatever.



```
# time: 2014-06-27 12:05:25 Pacific Daylight Time
# mode: julia
    println("ok")
# time: 2014-06-27 12:10:40 Pacific Daylight Time
# mode: julia
    function f(x)
    y = 2x + 7
    println(y)
    end
# time: 2014-06-27 12:11:15 Pacific Daylight Time
# mode: julia
    function f(x)
        y = 2x + 7
        println(y)
        end
# time: 2014-06-27 14:06:02 Pacific Daylight Time
# mode: julia
    1+1
```

To paste code directly into your command-prompt window you will have to right-click and select “paste”; key bindings do not work there.

Practice Problem 2



2. Write a function info(a, b) that takes a point (a, b) and returns three things about the segment from the origin to the point:

a) Its slope

b) Its midpoint

c) Its length (distance)

Output must be in the form “the slope from the origin is ____ ,” etc.

Write at least one line of documentation, *test your code*, and submit your code as a digital plain-text file.