**Basics of Julia for 15.053**

Julia Version 0.4.6

This document outlines some of the basic programming concepts and Julia syntax used in 15.053. It is ***not*** a comprehensive guide to the Julia programming language. If you have little to no experience with programming, you may find this document helpful.

**Common Types**

A **type** is a classification that identifies a type of data. It determines the possible values for that type, as well as the different operation that can be done on values of that type.

**Int** an integer

**Float** a number of fractional value, decimal

**String** a sequence of characters denoted by surrounding quotation marks " "

**Booleans** true or false

**Tuples** an ordered sequence of elements denoted by parentheses ( )

For more documentation on Types see

<http://docs.julialang.org/en/release-0.4/manual/integers-and-floating-point-numbers/>

<http://docs.julialang.org/en/release-0.4/manual/strings/>

**Mathematical Operations**

Below is a list of common mathematical operations that you can perform on numerical types.

**x + y** performs addition

**x – y** performs subtraction

**x \* y** performs multiplication

**x / y** performs division

**x ^ y** raises **x** to the **y**th power

**x % y** gives remainder when you divide **x** by **y**, same as **rem(x,y)**

**x = y** assigns the variable named "**x**" to the value **y**

**x == y** evaluates to a Boolean, **true** if **x** equals y, **false** otherwise

**x != y** evaluates to a Boolean, **true** if **x** does not equal **y**, **false** otherwise

**x > y** evaluates to a Boolean, **true** if **x** is greater than **y**, **false** otherwise

**x < y** evaluates to a Boolean, **true** if **x** is less than **y**, **false** otherwise

**x <= y** evaluates to a Boolean, **true** if **x** is less than or equal to **y**, **false** otherwise

**x >= y** evaluates to a Boolean, **true** if **x** is greater than or equal to **y**, **false** otherwise

**x += y** adds **y** to **x**, then reassigns **x** to the new value

**x -= y** subtracts **y** from **x**, then reassigns **x** to the new value

For more documentation on Mathematical Operations see

<http://docs.julialang.org/en/release-0.4/manual/mathematical-operations/>

**Printing Output**

Below we give two simple ways of printing program output.

**print("sum = ", 10)** prints **sum = 10**

**println("sum = ", 10)** prints **sum = 10** then skips a line

**Variables**

**Variables** are a way of assigning names to objects. Objects can be numbers, arrays, functions, and other data structures. Variable names can be anything, except for reserved keywords (for example, the words *in*, *for*, *while*, and *else* are reserved words). Variable names cannot start with a number.

Below are examples of variable instantiations.

**a = 4**  creates a variable called **a** that is associated with the value "4"

**x = [1 2 3]** creates a variable called **x** that is associated with a 1D-array of three elements

**y = [1 2 3; 4 5 6]** creates a variable called **y** that is associated with a 2x3-array of six

elements

**s = ["hi" "bye"]** creates a variable called **s** that is associated with a 1D-array of strings

**bol = false** creates a variable called **bol** that is associated with the Boolean of value

false

For more documentation on Variables see <http://docs.julialang.org/en/release-0.4/manual/variables/>

**Arrays, Indexing, and Methods**

**Creating Arrays**

Arrays are great structures for storing lists or matrices of information. They can be one-dimensional or multidimensional.

For example,

**x = [2 4 5 8 12]**  creates a variable called **x** that is associated with a 1D array of 5

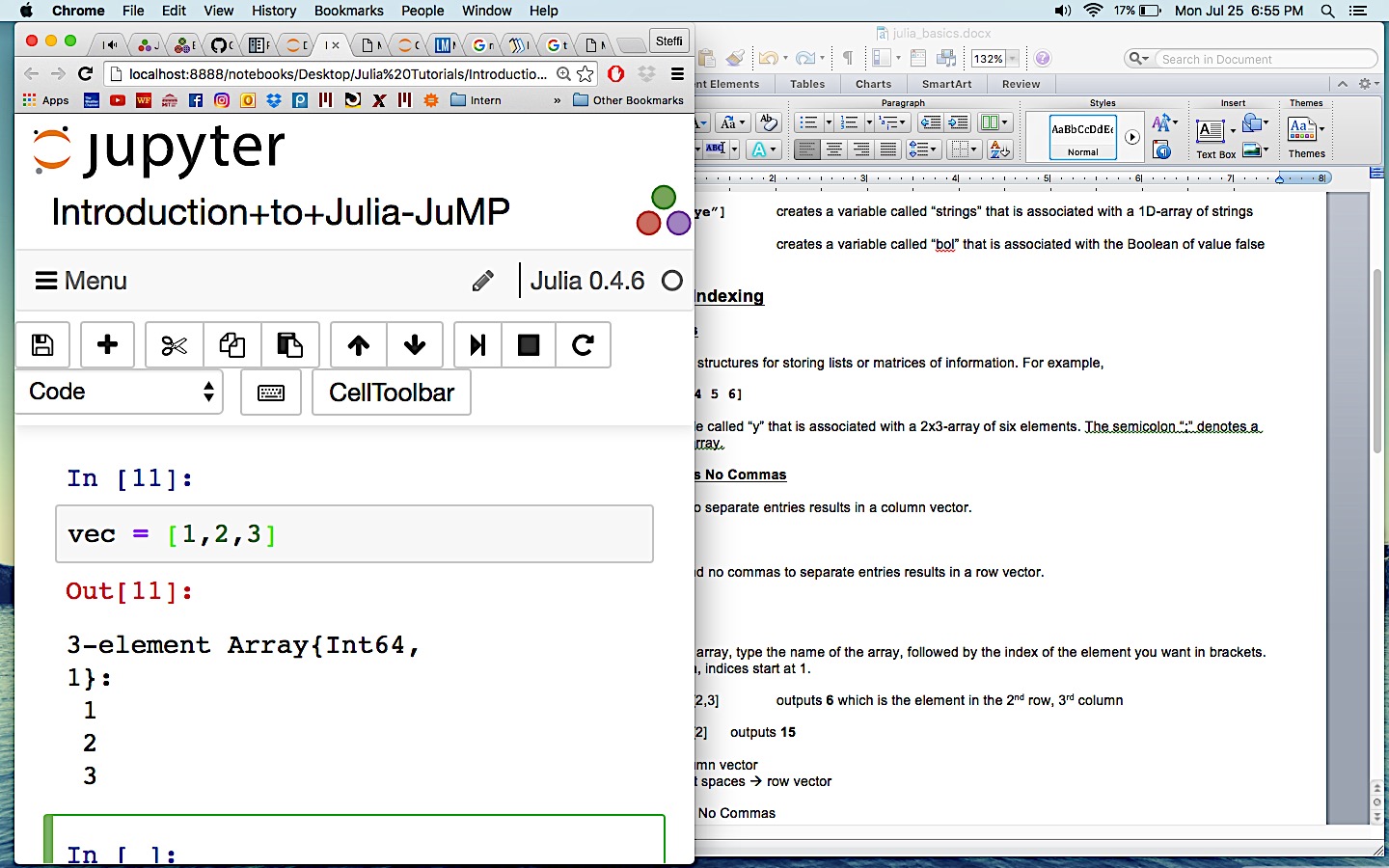
elements.

**y = [1 2 3; 4 5 6]** creates a variable called **y** that is associated with a 2x3-array of

six elements. The semicolon ";" denotes a new row of the array.

**Commas vs. No Commas in Arrays**

Using commas to separate entries results in a column vector.



Using spaces and no commas to separate entries results in a row vector.



**Indexing**

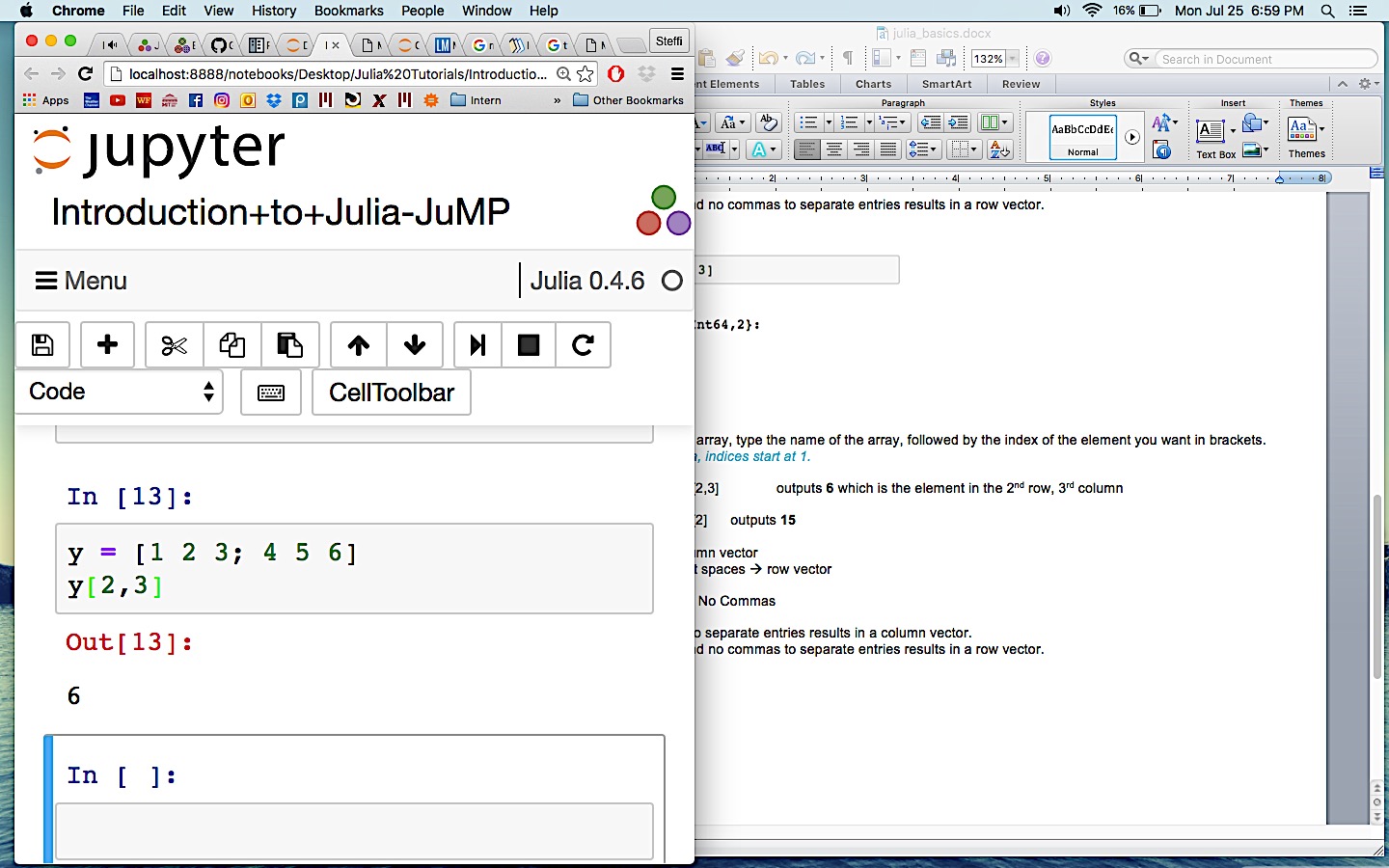
**Indexing** is used to specify the elements of an array. Indexing also allows you to get out certain bits of information from an array. To index into an array, type the name of the array, followed by the index of the element you want in brackets. *Note that in Julia, indices start at 1.*

Below, we index into the array named **x** to get out the third element, **3.2**.



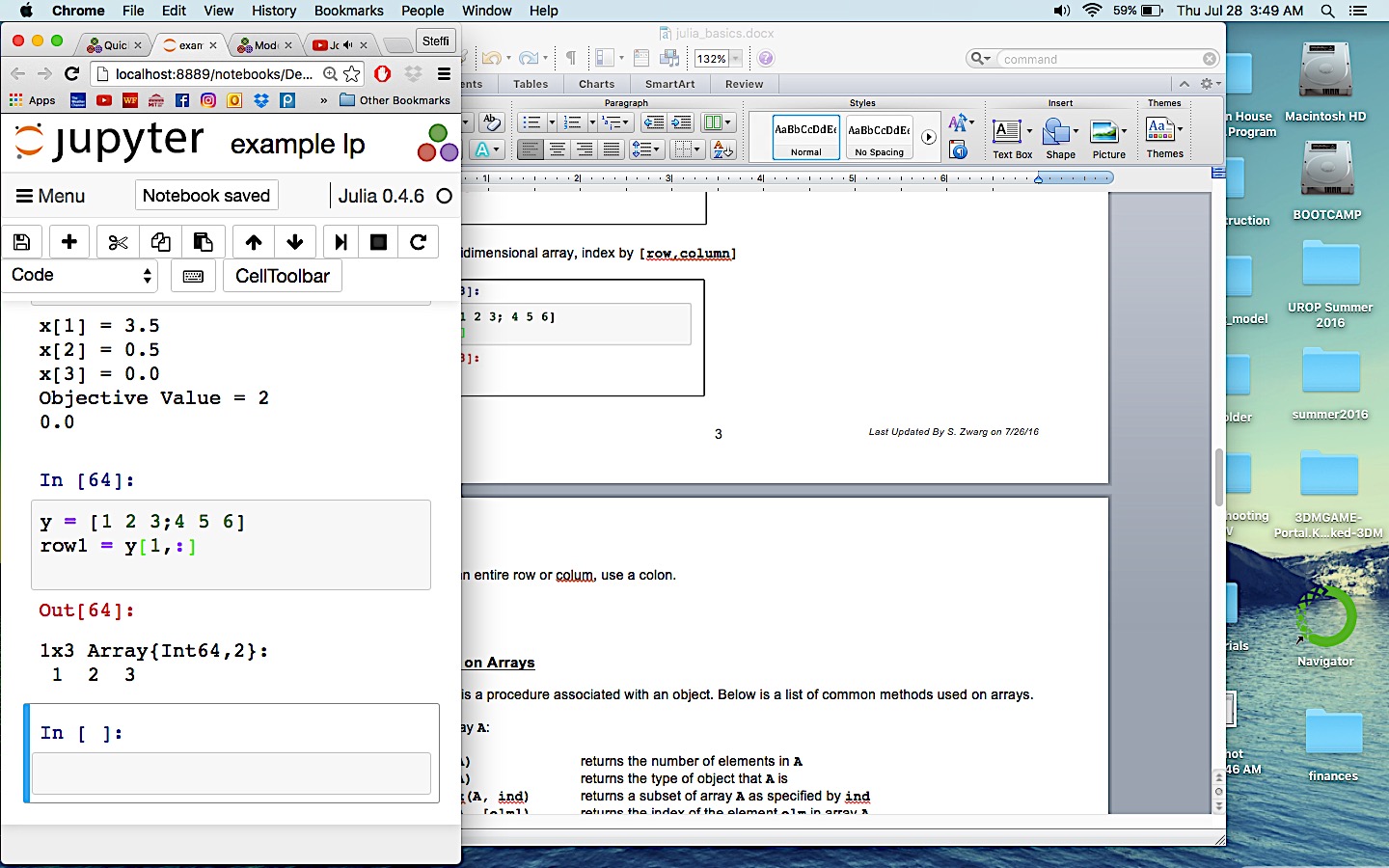
For a multidimensional array, index by **[row,column]**

Below we index into the named **y** to get out the element in the second row, third column, **6**.

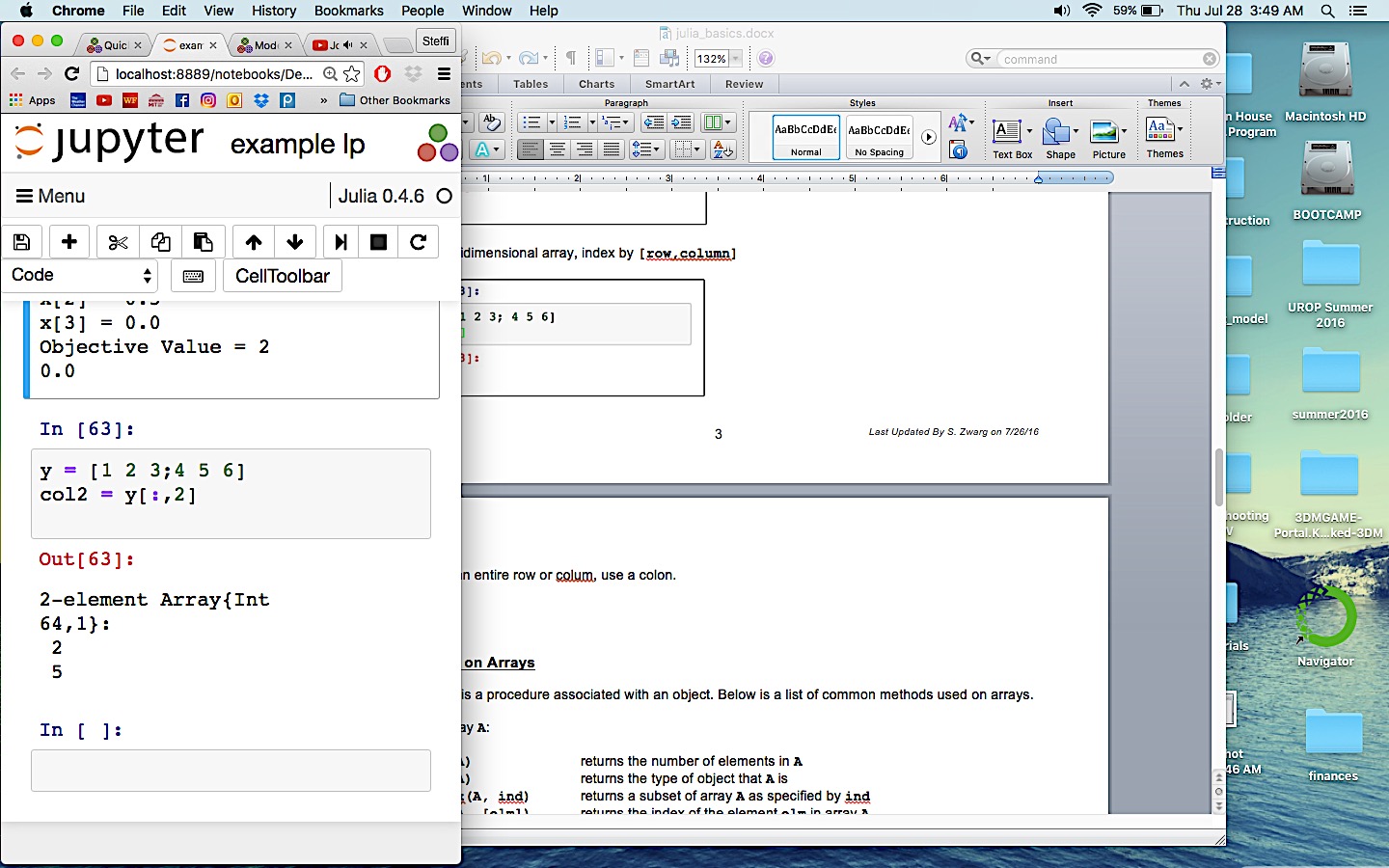


To index an entire row or column, use a colon.

Below, we index the entire first row of the array named **y**.



Below, we index the entire second column of the array named **y**.

****

**Methods on Arrays**

A **method** is a procedure associated with an object. Below is a list of common methods used on arrays.

For an array **A**:

**length(A)** returns the number of elements in **A**

**typeof(A)** returns the type of object that **A** is

**getindex(A, ind)** returns a subset of array **A** as specified by **ind**

**findin(A, [elm])** returns the index of the element **elm** in array **A**

**cat(dims, A, B)** concatenates **A** and **B** along the specified dimension **dims**

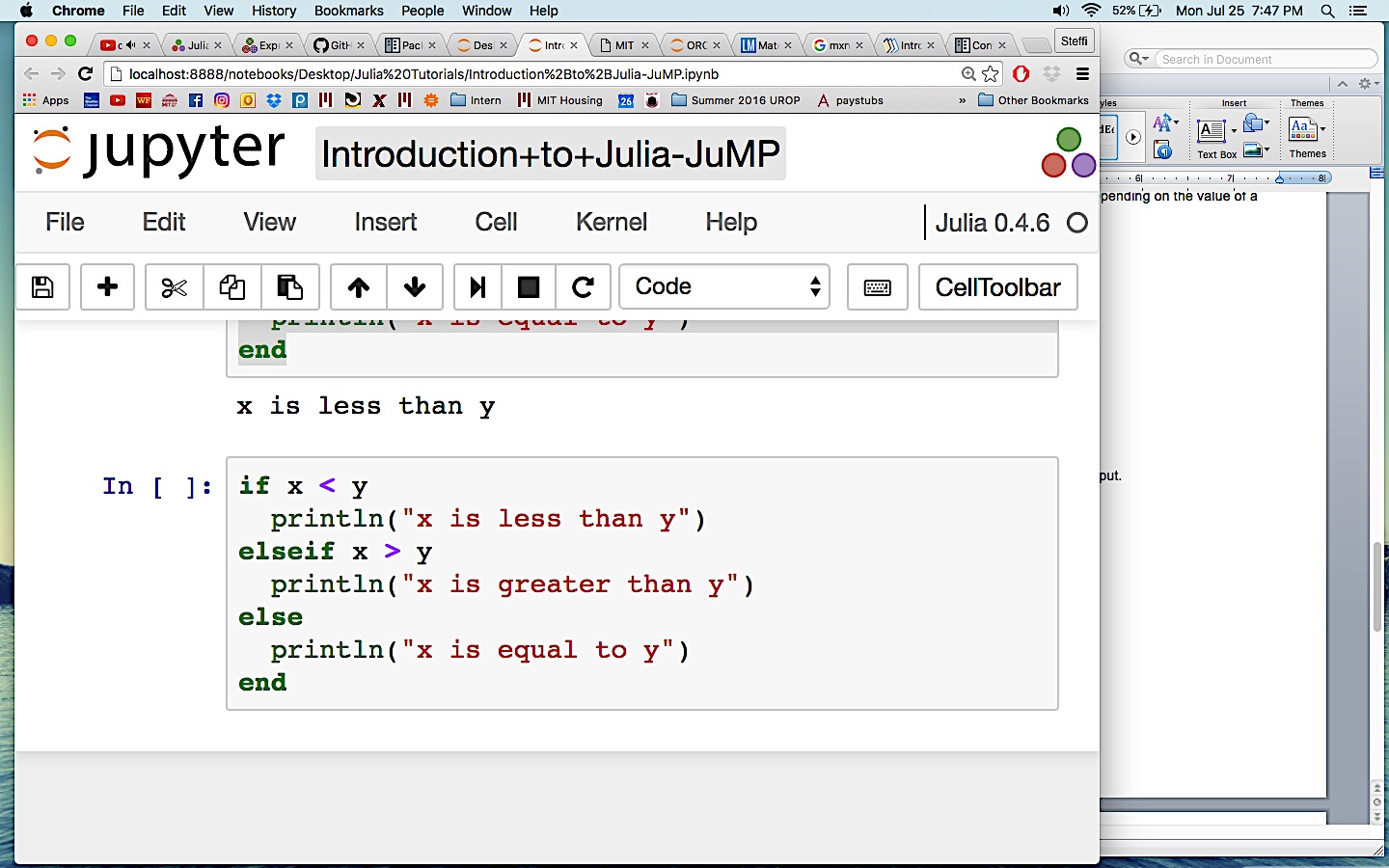
For more documentation on Arrays see <http://docs.julialang.org/en/release-0.4/stdlib/arrays/?highlight=array>

**Control Flow**

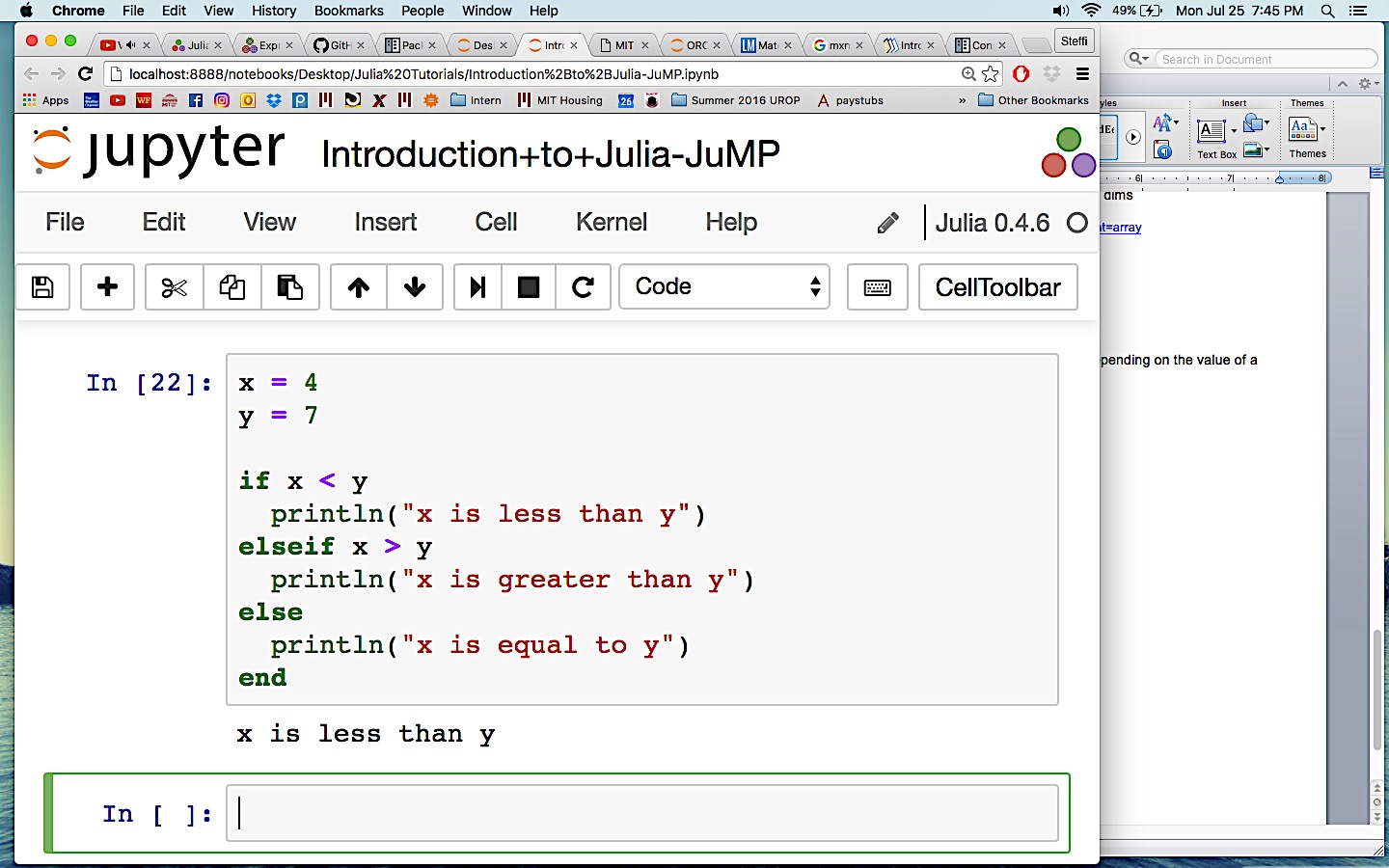
**if/elseif/else**

**Conditional evaluation** allows portions of code to be evaluated or not evaluated depending on the value of a Boolean expression. You do not need all if/elseif/else statements. You can have conditional evaluations with just an **if**, or just an **if/else**.

The general structure of conditional evaluation is as follows.



After assigning values to x and y and running the code we obtain the following output.



**Boolean Chains**

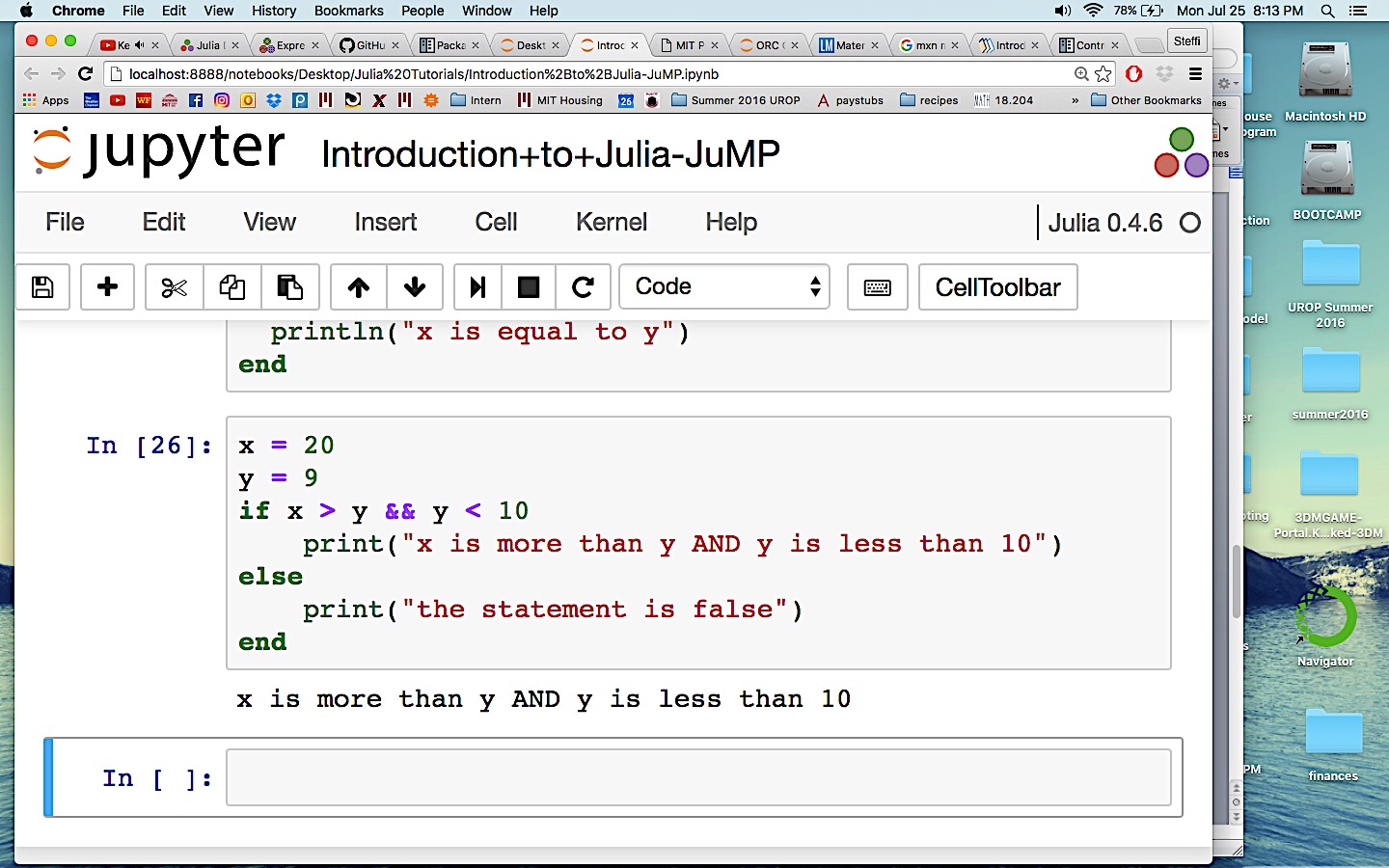
In Boolean chains, only the minimum number of expressions are evaluated as are necessary to determine the final Boolean value of the entire chain.

**a && b** the sub-expression **b** is only evaluated if **a** evaluates to **true**

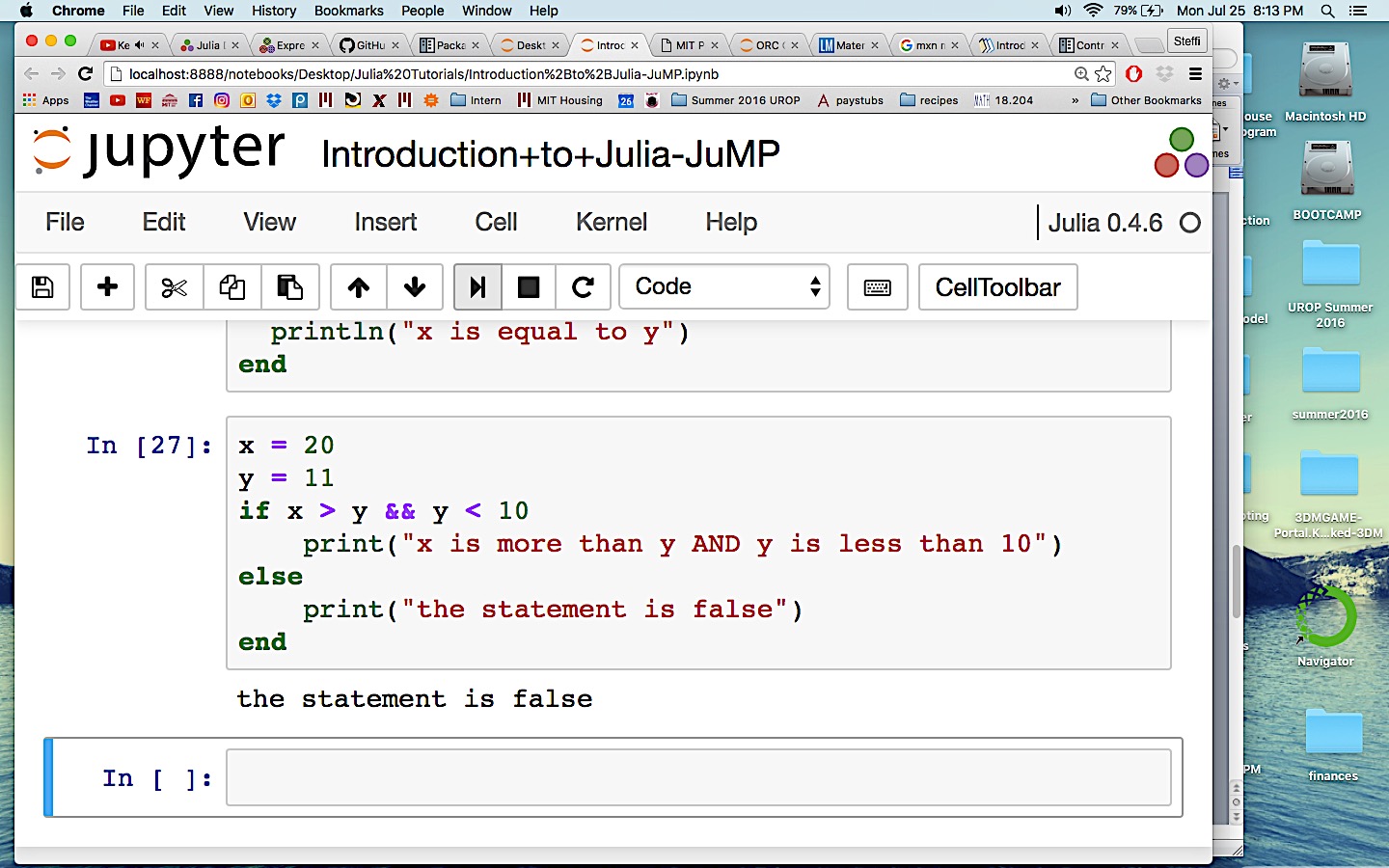
**a || b** the sub-expression **b** is only evaluated if **a** evaluates to **false**

Take a look at the different evaluation paths for different values of x and y in the following example which uses an **a && b** Boolean chain.

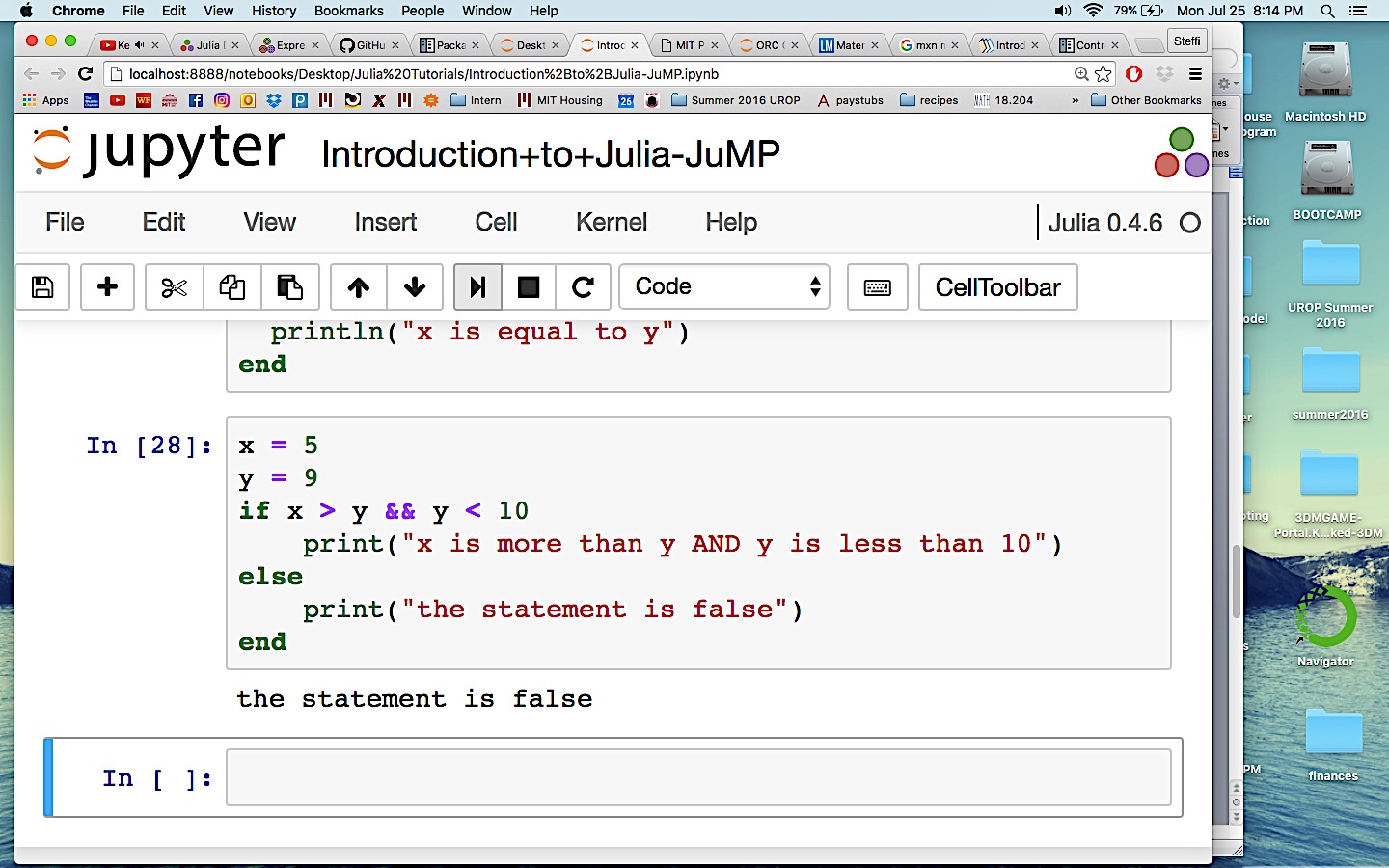
In **In [26]** The first part of the **if** statement evaluates to **true**, therefore the second part of the if statement is evaluated. The second part of the **if** statement evaluates to **true**, thus the **else** clause is not evaluated.



In **In [27]** the first part of the **if** statement evaluates to **true**, but the second part evaluates to **false**, thus the **else** clause is evaluated.



In **In [27]** the first part of the **if** statement evaluates to **false**, therefore the second part of the **if** statement is not evaluated and instead the **else** clause is evaluated.



**For Loops**

A **for** loop allows you to specify the number of iterations for the repeated execution of a code block. They are great when you know how many iterations you want to run.

The general form of a **for** loop is shown below. The example shows a **for** loop that calculates the sum of the integers 1 through 10 and prints the final result.

Note that to obtain a **range** of integers, we use the colon : symbol.



For more documentation on Control Flow see <http://docs.julialang.org/en/release-0.4/manual/control-flow/>

**Comments**

Comments in Julia are denoted by the pound sign **#**. Use comments to comment your code, separate blocks of code, etc.

**Release 0.4 Julia Documentation**

For the official Julia documentation for Julia release 0.4 see <http://docs.julialang.org/en/release-0.4/#manual>