0104 Variables

```
2+2
ans
myvar = 2+2
ans
myvar = myvar + 1
myvar += 1
aString = "testing"
ans
δ = 5 (\delta)
δ+7

#\alpha (don't assign so error is generated)
#ERROR: UndefVarError: α not defined

f\prime
\copyright
#http://docs.julialang.org/en/release-0.4/manual/unicode-input/

whos()

#Style guide: http://docs.julialang.org/en/release-0.4/manual/variables/
```

105 Strings

```
str = "hello world"
str[1]
str[11]
str[end]
str[end-10]
hello = "hello"
world = "world"
string(hello, " ", world)
newstr = string(hello, " ", world)
newstr
print("1+2 = ", 1 + 2)
"1 + 2 = $(1+2)"
```

106 Logical Operations

```
a = 1
1

julia> b = 2
2

julia> a < b
true

julia> a == b
false

julia> a > b

if a < b
    print(true)
    end
true
julia> if a < b
    print(true)
    else
    print(false)
    end
true

julia> if (a > b)
    print(true)
    else
    print(false)
    end
false

julia> if a > b
    print("a > b")
    elseif a < b
    print("a < b")
    else
    print("a == b")
    end

    result = begin
    a = 2
    b = 3
    a + b
    end
5
```

```
julia> result = a = 2; b = 3; a + b
```

107 Arrays

```
julia> [1,2,3,4]
4-element Array{Int64,1}:
 1
 2
 3
 4

julia> ["a", "b","c"]
3-element Array{ASCIIString,1}:
 "a"
 "b"
 "c"

julia> [1 2 3 4]
1x4 Array{Int64,2}:
 1  2  3  4

julia> [1 2 3; 4 5 6; 7 8 9]
3x3 Array{Int64,2}:
 1  2  3
 4  5  6
 7  8  9

julia> [1 2 3; 4 5 6; 8 9]
ERROR: ArgumentError: argument count does not match specified shape (expected 9,
got 8)
 in hvcat at /Applications/Julia-0.4.5.app/Contents/Resources/julia/lib/julia/sys.dylib

julia> array1 = [10, 20, 30]
3-element Array{Int64,1}:
 10
 20
 30

julia> sum(array1)
60

julia> sum([1,2,3])
6

julia> array2 = [1,2,3]
3-element Array{Int64,1}:
 1
 2
 3
```

```
julia> array1 .+ array2
3-element Array{Int64,1}:
 11
 22
 33

julia> array1 .* array2
3-element Array{Int64,1}:
 10
 40
 90

julia> array1 + array2
3-element Array{Int64,1}:
 11
 22
 33

 julia> array1[1]
10

julia> array1[0]
ERROR: BoundsError: attempt to access 3-element Array{Int64,1}:
 10
 20
 30
  at index [0]
 in getindex at array.jl:282

julia> array1[1] = 111
111

julia> array1[1]
111
julia> multiArray = [1 2 3; 4 5 6; 7 8 9]
3x3 Array{Int64,2}:
 1  2  3
 4  5  6
 7  8  9

julia> multiArray[2,3]
6

julia> multiArray[3,1]
7

julia> multiArray[:]
```

```
9-element Array{Int64,1}:
 1
 4
 7
 2
 5
 8
 3
 6
 9

julia> multiArray[:,2]
3-element Array{Int64,1}:
 2
 5
 8

 ulia> multiArray[1,:]
1x3 Array{Int64,2}:
 1  2  3

julia> multiArray[3,:]
1x3 Array{Int64,2}:
 7  8  9

 julia> multiArray[2,2:3]
1x2 Array{Int64,2}:
 5  6
```

108 Functions

```
julia> sqrt(25)

julia> arr = [1,4,2,3]
4-element Array{Int64,1}:

julia> sort(arr)

julia> function add(a,b)
      print("a=$a, b=$b")
      a + b
      end
add (generic function with 1 method)

julia> add(1,3)
a=1, b=34

#use println for better output
julia> function add(a,b)
      println("a=$a, b=$b")
      a + b
      end
add (generic function with 1 method)

julia> add(1,3)
a=1, b=3
4

julia> ∑(a,b) = a + b
∑ (generic function with 1 method)

julia> ∑(1,3)
4

#pass around function
julia> myfunc = add
add (generic function with 1 method)

julia> myfunc(1,5)
a=1, b=5
6
```

109 Functions II

```
julia> function daily_int(x)
       return x * .0056
       x + 1
       end
daily_int (generic function with 1 method)

julia> daily_int(5)
0.028

julia> I=.0026
0.0026

julia> function daily_int(x)
       if I>.005
          return x * .0056
       elseif I < .005
          return 0
       end
       end
daily_int (generic function with 1 method)

julia> daily_int(5)
0

julia> I=.0057
0.0057

julia> daily_int(5)
0.028

julia> x -> x^2
(anonymous function)

julia> function(x)
       x^2
       end
(anonymous function)

julia> x(5)
ERROR: UndefVarError: f not defined

julia> map(sqrt, [2,4,5])
julia> map(x->x^2,[2,4,5])

julia> printit(a,b...)=println("a=$a"),println("b=$b")
```

```
julia> function di(x,int=.0056)
       return x * int
       end

#keyword arguments
julia> function register_employee(fn, ln;dept="sales", salary_type="commission")
       print("fn=$fn, ln=$ln")
       end

julia> function register_employee(fn, ln;dept="sales", salary_type="commission")
       print("fn=$fn, ln=$ln, dept=$dept, salary_type=$salary_type")
       end
julia> register_employee("john", "doe")
julia> register_employee("john", "doe", dept="HR")

julia> register_employee("mike", "jones")
```

110 Functions III

```
#functions part 3

julia> function varargs(args...)
      return args
      end
varargs (generic function with 1 method)

julia> varargs(1,2,3)
(1,2,3)

julia> function sqrtit(args...)
      return map(x->x^2,args)
      end
sqrtit (generic function with 1 method)

julia> sqrtit(2,4,5)
(4,16,25)

#weight on planet
#weight calculator: http://www.learningaboutelectronics.com/Articles/Weight-on-saturn-conversion-calculator.php#answer
#
julia> γ=9.81
9.81

julia> planet_weight(lbs, mass) = lbs/γ*mass
planet_weight (generic function with 1 method)

julia> planet_weight(160,24.79)
404.322120285423

#credit card interest

julia> cc_int(prin,int,days) = prin*int*days
cc_int (generic function with 1 method)

#.041%
julia> cc_int(633,.00041,30)
7.7859
```

111 Types

```
#types

julia> mystring = "test"
"test"

julia> typeof(mystring)
ASCIIString

julia> typeof(5)
Int64

julia> myint = 5
5

julia> typeof(myint)
Int64

julia> typeof(Int64)
DataType

julia> super(Int64)
Signed

julia> super(Signed)
Integer

julia> super(Integer)
Real

julia> super(Real)
Number

julia> super(Number)
Any

julia> super(Any)
Any

#no more subtypes
julia> subtypes(Int64)
0-element Array{Any,1}

julia> subtypes(Number)

#custom type
julia> type Person
```

```
    first_name::ASCIIString
    last_name::ASCIIString
    end

julia> person1 = Person("john","doe")
Person("john","doe")

julia> typeof(person1)
Person


julia> subtypes(Real)
```

112 Dictionaries

```
julia> mydictionary = Dict("car1"=>"2door", "car2"=>"4door", "car3"=>"3door")
Dict{ASCIIString,ASCIIString} with 3 entries:

julia> mydictionary["car1"]

julia> keys(mydictionary)

julia> values(mydictionary)

julia> mydict["car1"]="5door"
"5door"

julia> mydict

julia> haskey(mydict,"car1")

julia> in(("car1" => "2door"), mydict)
false

julia> in(("car1" => "5door"), mydict)
true

julia> mydict["car4"] = "2door"
"2door"

julia> mydict

julia> mydictionary2 = Dict()

julia> mydictionary2["item1"] = "value1"
"value1"

julia> mydictionary2
```

113 Loops

```
for i = ["a", "b", "c"]
      println(i)
      end

julia> for car=["2door", "4door", "3door"]
      println("# of doors = $car")
      end

julia> for car in ["2door", "4door", "3door"]
      println("# of doors = $car")
      end

julia> typeof(car)
ERROR: UndefVarError: car not defined

julia> carArray = ["2door", "4door", "3door"]
3-element Array{ASCIIString,1}:
=

julia> typeof(carArray)
Array{ASCIIString,1}

julia> for c = Dict("car1"=>"2door", "car2"=>"4door", "car3"=>"3door")
      println("$c[1] is a $c[2]")
      end

#fix output like this
julia> for c = Dict("car1"=>"2door", "car2"=>"4door", "car3"=>"3door")
          println("$(c[1]) is a $(c[2])")
          end

#note the type is Dict{ASCIIString,ASCIIString}. We must access elements using strings.

julia> for k,v = Dict("car1"=>"2door", "car2"=>"4door", "car3"=>"3door")
          println("$k is a $v")
          end
ERROR: syntax: invalid iteration specification

#we need a tuple
julia> for (k,v) = Dict("car1"=>"2door", "car2"=>"4door", "car3"=>"3door")
          println("$k is a $v")
          end

#can also use "in" keyword
julia> for (k,v) in Dict("car1"=>"2door", "car2"=>"4door", "car3"=>"3door")
          println("$k is a $v")
```

```
        end

#while loop
julia> a = [1,2,3,4,5]
5-element Array{Int64,1}:

julia> i = 1
1

julia> while i < 4
    println(a[i])
    i += 1
    End
```

114 List Comprehensions

```
julia> squared = Array(Int64, 0)
0-element Array{Int64,1}

julia> for x in 1:5
       push!(squared,x^2)
       end

julia> squared
5-element Array{Int64,1}:
  1
  4
  9
 16
 25

julia> squared2 = [x^2 for x in 1:5]
5-element Array{Int64,1}:
  1
  4
  9
 16
 25

 julia> filter(x -> x>5, [3,4,5,6,7])
2-element Array{Int64,1}:
 6
 7

julia> filter(x -> x % 2 == 0, squared)
2-element Array{Int64,1}:
  4
 16

julia> filter(x -> x % 2 == 0, [1,2,3,4,5])
2-element Array{Int64,1}:
 2
 4
```

115 Scope

```
julia> func1(x) = x*2
func1 (generic function with 1 method)

julia> x
ERROR: UndefVarError: x not defined

julia> γ=4
4

julia> γ
4

julia> func2(x) = γ * x
func2 (generic function with 1 method)

julia> func2(2)
8

julia> function func3(x)
      y = 5
      print(x*y)
      end
func3 (generic function with 1 method)

julia> func3(3)
15
julia> y
ERROR: UndefVarError: y not defined
```

116 Modules

```
julia> module MyModule
       export bar
       bar(x) = 2x
       end

julia> bar(2)
ERROR: UndefVarError: bar not defined

julia> MyModule.bar(2)
4

julia> using MyModule

julia> bar(4)
8

julia> module MyModule2
       square(x) = x^2
       baragain(x) = bar(x)
       end
MyModule2

julia> using MyModule2

julia> square(2)
ERROR: UndefVarError: square not defined

module MyModule2
       using MyModule
       export baragain
       baragain() = print(bar(2))
       end

julia> MyModule2.baragain()
```

201 File IO - Reading

```
julia> pwd()
"/Users/brettromero"

julia> homedir()
"/Users/brettromero"

julia> readdir()

julia> f = open("juliatest.txt")

julia> s = readall(f)

julia> close(f)

#doesn't work if already read content
julia> f = open("juliatest.txt")
IOStream(<file juliatest.txt>)

julia> s = readall(f)
"This is line one\nThis is line two\nThis is line three"

julia> lines = readlines(f)
0-element Array{ByteString,1}

julia> close(f)

#notice file is closed but still have access to its contents in variable
julia> for l in lines
    println("$i: $l")
    i += 1
    end

julia> f = open("juliatest.txt")
IOStream(<file juliatest.txt>)

julia> for ln in eachline(f)
    println("$(length(ln)), $ln")
    end

julia> close(f)

julia> f = open("juliatest.txt")
IOStream(<file juliatest.txt>)

julia> i = 1
1
```

```
julia> while !eof(f)
       line = readline(f)
       println("$i: $line")
       i += 1
       end

#changing directories
julia> cd

julia> readdir("go/src")

julia> cd("go/src")

julia> pwd()

julia> cd(homedir())

julia> pwd()

julia> go_path = joinpath(pwd(), "go/src")

julia> f = open(joinpath(go_path, "juliatest.txt"))

julia> lines = readlines(f)

julia> close(f)

#stat fields: http://docs.julialang.org/en/release-0.4/stdlib/file/#Base.stat

julia> stat("juliatest.txt")
StatStruct(mode=100644, size=52)

julia> stats = stat("juliatest.txt")
StatStruct(mode=100644, size=52)

julia> stats.device
0x0000000001000004

julia> stats.uid
0x00000000000001f6

julia> stats.ctime

julia> for n in fieldnames(stat(i))
```

202 File IO - Working with Directories

```
#changing directories
julia> cd

julia> readdir("go/src")

julia> cd("go/src")

julia> pwd()

julia> cd(homedir())

julia> pwd()

julia> go_path = joinpath(pwd(), "go/src")

julia> f = open(joinpath(go_path, "juliatest.txt"))

julia> lines = readlines(f)

julia> close(f)

#stat fields: http://docs.julialang.org/en/release-0.4/stdlib/file/#Base.stat

julia> stat("juliatest.txt")
StatStruct(mode=100644, size=52)

julia> stats = stat("juliatest.txt")
StatStruct(mode=100644, size=52)

julia> stats.device
0x0000000001000004

julia> stats.uid
0x00000000000001f6

julia> stats.ctime

julia> for n in fieldnames(stat(i))
```

203 File IO - Writing to Files

```
julia> pwd()
"/Users/brettromero"

julia> f = open("juliawrite.txt", "w")
IOStream(<file juliawrite.txt>)

julia> write(f, "line 1\n")
7

julia> close(f)

julia> f = open("juliawrite.txt", "w")
IOStream(<file juliawrite.txt>)

julia> write(f, "line 1\n")
7

julia> write(f, "line 2\n")
7

julia> write(f, "line 3\n")
7

julia> close(f)

julia> f = open("juliawrite.txt")
IOStream(<file juliawrite.txt>)

julia> s = readall(f)
"line 1\nline 2\nline 3\n"

julia> close(f)
```

204 Macros

#macros

205 Meta Programming and Symbols

#meta programming

```
julia> a_prog = "5 + 5"
"5 + 5"

julia> express1 = parse(a_prog)
:(5 + 5)

julia> typeof(express1)
Expr

julia> express1.head
:call

julia> express1.args

julia> express1.typ
Any

#prefix notation
julia> express2 = Expr(:call, :+, 1, 1)
:(1 + 1)

julia> express2 = Expr(:call, :+, 5, 5)
:(5 + 5)

julia> express1 == express2
true

julia> dump(express1)
Expr
  head: Symbol call
  args: Array(Any,(3,))
    1: Symbol +
    2: Int64 5
    3: Int64 5
  typ: Any

julia> Meta.show_sexpr(express1)
(:call, :+, 5, 5)

#symbols
julia> :myfunc
:myfunc
```

```
julia> typeof(ans)
Symbol

#symbol() function
julia> :myfunc == symbol("myfunc")
true

julia> symbol("myfunc", 5)
:myfunc5

julia> symbol(:myfunc, '-', "adder")
symbol("myfunc-adder")
```

206 Performance Tips

```
#performance tips
julia> const DEFAULT_VAL = 0
0
julia> typeof(DEFAULT_VAL)
Int64

julia> global x = 5.0
5.0

julia> y(a,b) = a + 1
y (generic function with 1 method)

julia> y(1,1)
2

julia> y(a::Int, b::Int) = a + b
y (generic function with 2 methods)

julia> y(1,1)
2

julia> y(1,5.0)
2

julia> y(1,5.1)
2

julia> z(a::Int, b::Number) = a + b
z (generic function with 1 method)

julia> z(1,5.0)
6.0

julia> z(1,2)
3

julia> z(1,4.5)
5.5

julia> function c(b)
      b * 400^700
      end
c (generic function with 1 method)

julia> @time c(10)
```

```
julia> @time c(10*1000)
  0.000663 seconds (5 allocations: 176 bytes)
0

julia> @time c(10*1000)
  0.000004 seconds (5 allocations: 176 bytes)
0

julia> @time c(10)
  0.000003 seconds (4 allocations: 160 bytes)
0
```

207 Error Handling I

```
#exception handling

julia> sqrt(-5)

julia> my_sqrt(x)
ERROR: UndefVarError: my_sqrt not defined

julia> error("nope!")

julia> function my_sqrt(x)
      if x < 1
        error("x must be greater than 1")
      end
      sqrt(x)
      end

my_sqrt (generic function with 1 method)

julia> my_sqrt(1)
1.0

julia> my_sqrt(0)

julia> my_sqrt(-1)

julia> function my_sqrt2(x)
      try
        sqrt(x)
      catch e
        println(e)
      end
      end

my_sqrt2 (generic function with 1 method)

julia> my_sqrt2(4)
2.0

julia> my_sqrt2(0)
0.0

julia> my_sqrt2(-1)
DomainError()

julia> function my_sqrt2(x)
      try
```

```
        sqrt(x)
      catch e
        println(string("An error occurred: ", e))
      end
      end
my_sqrt2 (generic function with 1 method)

julia> my_sqrt2(-1)
An error occurred DomainError()
```

208 Error Handling II

```
#error handling 2

julia> try 5+5 catch ex end
10

julia> try sqrt(-1)
       catch
         a
         println(a)
       end
ERROR: UndefVarError: a not defined

julia> try sqrt(-1)
       catch a
         println(a)
       end
DomainError()

julia> try 5+5 catch ex end
10

julia> try sqrt(-1) catch ex end

julia> try sqrt(-1) catch ex println(ex) end
DomainError()

#info, warn, error

julia> error("my error")
ERROR: my error
 in error at /Applications/Julia-0.4.5.app/Contents/Resources/julia/lib/julia/sys.dylib

julia> info("just an info")
INFO: just an info

julia> 1+4;info("just an info")
INFO: just an info

julia> info("just an info");1+4
INFO: just an info
5

julia> warn("just a warning");1+4
WARNING: just a warning
5
```

```
#finally block

julia> f = open("juliatest.txt")
IOStream(<file juliatest.txt>)

julia> isopen(f)
true

julia> try
         sqrt(-1)
       finally
         close(f)
       end
ERROR: DomainError:
 [inlined code] from none:2
 in anonymous at no file:0

julia> isopen(f)
false

julia> f = open("juliatest.txt")
IOStream(<file juliatest.txt>)

julia> isopen(f)
true

julia> try
         sqrt(-1)
       catch e
         println(e)
       finally
         close(f)
       end
DomainError()
```

209 Multiple Dispatch

#multiple dispatch

```
julia> func(a, b) = "base"
func (generic function with 1 method)

julia> func(a::Number, b::Number) = "a and b are numbers"
func (generic function with 2 methods)

julia> func(a::Number, b) = "a=number, b=?"

func (generic function with 3 methods)

julia> func(a, b::Number) = "a=?, b=number"
func (generic function with 4 methods)

julia> func(a::Integer, b::Integer) = "a and b are integers"
func (generic function with 5 methods)

julia> func(5.0, 1)
"a and b are numbers"

julia> func(5,"something")
"a=number, b=?"

julia> func(5,10)
"a and b are integers"

julia> func("something", [2,4])
"base"

julia> @which func(5.0, 1)
func(a::Number, b::Number) at none:1

julia> @which func(5, "something")
func(a::Number, b) at none:1

julia> @which func(5,10)
func(a::Integer, b::Integer) at none:1
```