CS 696 Intro to Big Data: Tools and Methods
Fall Semester, 2016
Doc 7 Julia Introduction
Sep 15, 2016

Nesting Functions

```
sinc(tan(sqrt(sin(13))))
```

```
13 |> sin |> sqrt |> tan |> sinc
```

```
a = sin(13)
b = sqrt(a)
c = tan(b)
sinc(c)
```

Lazy.jl

```
using Lazy

@> 13 sin sqrt tan sinc
```

Sends value to first argument
of next function

```
@> x f = f(x)
@> x g f == f(g(x))
@> x a b c d e == e(d(c(b(a(x)))))

@> x g(y, z) f == f(g(x, y, z))

@> x g f(y, z) == f(g(x), y, z)
```

```
@> [1 2; 3 4] (+) ([1 1;1 1]) (*)([1,2])


@> [1 2; 3 4] begin
  +([1 1;1 1])
  *([1,2])
end
```

Lazy.jl @>>

Sends value to last argument of next function

@>> x g(1, 2) f == f(g(1, 2, x))

@>> x g f(1, 2) == f(1, 2, g(x))

@>> x g f(y, z) h(w) == h(x, f(y, z, g(x)))

```
@as x [1 2; 3 4] begin
  x + [1 1; 1 1]
  x * [1,2]
end
```

Lazy.jl @as

Define variable that takes on the result of previous expression

```
@as x 5 f(x, 4) g(3, x) == g(3, f(x, 4))
```

```
@as x [1 2; 3 4] begin
  x + [1 1; 1 1]
  x * [1,2]
end
```

Unit Testing

Testing

Johnson's Law

If it is not tested it does not work

The more time between coding and testing

- More effort is needed to write tests
- More effort is needed to find bugs
- Fewer bugs are found
- Time is wasted working with buggy code
- Development time increases
- Quality decreases

Unit Testing

Tests individual code segments

Automated tests

What wrong with:

Using print statements

Writing driver program in main

Writing small sample programs to run code

Running program and testing it be using it

We have a QA Team, so why should I write tests?

When to Write Tests

First write the tests

Then write the code to be tested

Writing tests first saves time

- Makes you clear of the interface & functionality of the code
- Removes temptation to skip tests

What to Test

Everything that could possibly break

Test values

- Inside valid range
- Outside valid range
- On the boundary between valid/invalid

GUIs are very hard to test

- Keep GUI layer very thin
- Unit test program behind the GUI, not the GUI

Common Things Programs Handle Incorrectly

Adapted with permission from “A Short Catalog of Test Ideas” by Brian Marick,
http://www.testing.com/writings.html

Strings

Empty String

Collections

Empty Collection
Collection with one element
Collection with duplicate elements
Collections with maximum possible size

Numbers

Zero
The smallest number
Just below the smallest number
The largest number
Just above the largest number

```
using Base.Test

foo(x) = length(x)^2


@test foo("bar") == 9
@test foo("bar") == 10          test failed: 9 == 10

@test_throws MethodError foo(:foo)
@test_throws MethodError foo(“bar”)        test failed

@test_approx_eq 1. 0.999999999999
@test_approx_eq_eps 1. 0.999 1e-2


@testset "Foo Tests" begin                # Julia 0.5
        @test foo("a")   == 1
        @test foo("ab")  == 4
        @test foo("abc") == 9
end
```

Performance Issues

First Run Compiles Code

```
function sumer(n)
  localsum = 0
  for k in 1:n
    localsum = k + localsum
  end
  localsum
end
```

```
@time sumer(1_000_000)          0.002106 seconds (769 allocations: 44.500 KB)

@time sumer(1_000_000)          0.000003 seconds (5 allocations: 176 bytes)
```

Run code on small input to compile it
Then run on large input

Use Functions - Top level treated differently

```
const N = 1_000_000
sum = 0
@time  for k = 1:N
  sum = sum + k
end
```

Time	Memory
0.05 sec	30.5 MB

```
function sumer(n)
  localsum = 0
  for k in 1:n
    localsum = k + localsum
  end
  localsum
end

@time sumer(N)
```

Time	Memory
0.000003 sec	176 bytes

Avoid changing the type of a variable

```
function sumerbad(n)
  localsum = 0          # Int
  for k in 1:0.5:n÷2
    localsum = k + localsum # now float
  end
  localsum
end
```

@time sumerbad(N)

Time	Memory
0.058 sec	30.5 MB

```
function sumergood(n)
  localsum = 0.0        # float
  for k in 1:0.5:n÷2
    localsum = k + localsum # float
  end
  localsum
end
```

@time sumergood(N)

Time	Memory
0.004 sec	176 bytes

Type Stability

Return type of a function should only depend on the type of its arguments

Compiler can generate effictient code for type stable functions

pos(x) = x < 0 ? 0 : x

pos(2.3) returns a float
pos(-2.3) returns an integer

pos(x) = x < x ? zero(x) : x

pos(2.3) returns a float
pos(-2.3) returns a float
Type stable
25% faster

Testing for Type Stability @code_warntype

```
pos(x) = x < 0 ? 0 : x

@code_warntype pos(2.3)

Variables:
  x::Float64
  ##fy#13757::Float64

Body:
  begin # /Users/whitney/Courses/696/Fall 16/julia-examples/Scratch.jl, line 10:
      ##fy#13757 = (Base.box)(Float64,(Base.sitofp)(Float64,0))
      unless (Base.box)(Base.Bool,(Base.or_int)((Base.lt_float)(x::Float64,##fy#13757::Float64)::Bool,(Base.box)(Base.Bool,(Base.and_int)((Base.box)(Base.Bool,(Base.and_int)((Base.eq_float)(x::Float64,##fy#13757::Float64)::Bool,(Base.lt_float)(##fy#13757::Float64,9.223372036854776e18)::Bool)),(Base.slt_int)((Base.box)(Int64,(Base.fptosi)(In
0)::Bool)))) goto 0
      return 0
      0:
      return x::Float64
  end::UNION{FLOAT64,INT64}
```

Not type stable

Testing for Type Stability @code_warntype

```
pos(x) = x < zero(x) ? 0 : x
```

```
@code_warntype pos(2.3)
```

```
Variables:
  x::Float64
  ##fy#13757::Float64

Body:
  begin  # /Users/whitney/Courses/696/Fall 16/julia-examples/Scratch.jl, line 10:
      ##fy#13757 = (Base.box)(Float64,(Base.sitofp)(Float64,0))
      unless (Base.box)(Base.Bool,(Base.or_int)((Base.lt_float)(x::Float64,##fy#13757::Float64)::Bool,(Base.box)(Base.Bool,(Base.and_int)((Base.box)(Base.Bool,(Base.and_int)((Base.eq_float)(x::Float64,##fy#13757::Float64)::Bool,(Base.lt_float)(##fy#13757::Float64,9.223372036854776e18)::Bool)),(Base.slt_int)((Base.box)(Int64,(Base.fptosi)(In
0)::Bool)))) goto 0
          return 0
          0:
          return x::Float64
  end:: Float64
```

type stable

Better Benchmarking

@time can not measure small amounts of time accurately

```
Pkg.add("BenchmarkTools")
using BenchmarkTools

function sumer(n)
  localsum = 0.0
  for k in 1:0.5:n÷2
    localsum = k + localsum
  end
  localsum
end


@benchmark sumer(N)
```

```
BenchmarkTools.Trial:
  samples:          1423
  evals/sample:     1
  time tolerance:   5.00%
  memory tolerance: 1.00%
  memory estimate:  0.00 bytes
  allocs estimate:  0
  minimum time:     2.98 ms (0.00% GC)
  median time:      3.45 ms (0.00% GC)
  mean time:        3.50 ms (0.00% GC)
  maximum time:     6.11 ms (0.00% GC)
```

Profiling

```
using Base.Profile
function testfunc()
    x = rand(10000, 1000)
    y = std(x, 1)
    return y
end

@profile testfunc()
Profile.print()

Profile.clear()
```

Number of times seen by profiler

```
230 .../v0.4/Atom/src/eval.jl; anonymous; line: 61
 230 .../v0.4/Atom/src/eval.jl; withpath; line: 53
  julia/sys.dylib; abstract_eval_call; (unknown line)
                ...
               18 ...b/julia/sys.dylib; abstract_eval; (unknown line)
                18 ...b/julia/sys.dylib; abstract_eval_call; (unknown line)
                 18 ...b/julia/sys.dylib; abstract_call; (unknown line)
                  18 .../julia/sys.dylib; abstract_call_gf; (unknown line)
                   18 .../julia/sys.dylib; typeinf; (unknown line)
                    18 ...julia/sys.dylib; typeinf; (unknown line)
                     18 ...julia/sys.dylib; typeinf_uncached; (unknown line)
                      16 ...julia/sys.dylib; abstract_eval; (unknown line)
                       16 ...ulia/sys.dylib; abstract_eval_call; (unknown line)
```

Data

Data Formats

- CSV, TSV, WSV
- Excel, .xsl

Storing & Retrieving

- Files
- SQL databases
- No SQL databases
- New SQL databases
- Distributed
  - Files
  - Data structures
  - Databases

Data Manipulation

Data Cleaning

You have 230 GB of data

Your program has an I/O error reading the data

Now what?

80% the work of analyzing data is cleaning the data

Temperature Data - Daily Highs

City	6/7/1987	6/8/1987
San Diego	82	82
New York City	95	191
Timbuckto		65
New Deli	101	98

11,000 weather stations world wide
Hourly reporting
10 year history

11000 * 24 * 35 * 10 = 963_600_000

Data points just for temperature

How do you find and deal with errors?

80% the work of analyzing data is cleaning the data

Write programs to find errors

But you have to know what to look for

Write programs to correct errors

You have to know how to correct the error

80% the work of analyzing data is cleaning the data

Julia DataFrames

Similar to datafames in Python, R, Spark

Contains

NA - datatype for missing data

DataArray - Array that handles NA

DataFrame - tabular data sets (spreadsheet)

NA

```
false && NA == false
true || NA     == true

true && NA == NA
false || NA  == NA
1 + NA       == NA

NA && false         # Error
```

If an operation contains NA the result will be

NA if operation can't determine correct result

```
gender = DataFrame(ID = 1:4, Gender = ["M", "F", "F", "M"])
```

```
gender2 = DataFrame()
gender2[:ID] = 1:4
gender2[:Gender] = ["M", "F", "F", "M"]

gender2 == gender

gender[1,2] == "M"
gender[2,:Gender] == "F"
```

```
4×2 DataFrames.DataFrame
│ Row │ ID │ Gender │
├─────┼────┼────────┤
│ 1   │ 1  │ "M"    │
│ 2   │ 2  │ "F"    │
│ 3   │ 3  │ "F"    │
│ 4   │ 4  │ "M"    │
```

```
gender[1,2] == "M"
gender[2,:Gender] == "F"
gender[2] == gender[:Gender] == ["M", "F", "F", "M"]
gender[1] == gender[:ID] = [1,2,3,4]

gender[1:3,:]
```

```
│ Row │ ID │ Gender │
├─────┼────┼────────┤
│ 1   │ 1  │ "M"    │
│ 2   │ 2  │ "F"    │
│ 3   │ 3  │ "F"    │
```


Thursday, September 15, 16

All Columns in dataframe need to be the same length

```
gender = DataFrame(ID = 1:4, Gender = ["M", "F", "F", "M"])

mean(large[:A]) == 50.5
nrows = size(large,1) == 100
ncols = size(large,2) == 2

head(large)       # returns first 6 rows
tail(large)       # returns last 6 rows
```

```
describe(large)
```

```
A
Min      1.0
1st Qu.  25.75
Median   50.5
Mean     50.5
3rd Qu.  75.25
Max      100.0
NAs      0
NA%      0.0%

B
```

Reading/Writing CSV files

```
df = readtable("data.csv")

df = readtable("data.tsv")

df = readtable("data.wsv")

df = readtable("data.txt", separator = '\t')

df = readtable("data.txt", header = false)
```

```
df = DataFrame(A = 1:10)

writetable("output.csv", df)

writetable("output.dat", df, separator = ',', header = false)

writetable("output.dat", df, quotemark = '\"', separator = ',')

writetable("output.dat", df, header = false)
```

NA Example

naexample.csv

```
x,y
3,8
2,NA
,3
```

naexample = readtable("naexample.csv")

```
3×2 DataFrames.DataFrame
| Row | x  | y  |
|-----|----|----|
| 1   | 3  | 8  |
| 2   | 2  | NA |
| 3   | NA | 3  |
```

Missing values & NA
Become NA

mean(naexample[:y]) == NA

What to do with NA values?

mean(naexample[:y]) == NA

What does it make sense to do with the data?

- Find the correct value
- Use a default value
- Remove the row(s) or column(s) with NA

Finding NAs

naexample.csv

```
x,y
3,8
2,NA
,3
```

How to find if a column contains an NA

```
isna(naexample[:y])              # [false, true, false]
any(isna(naexample[:y]))         # true
```

Find the row(s) of the column that contains an NA

```
find(x -> isna(x),naexample[:y])      # [ 2 ]
```

Return a new Dataframe with rows removed that have NA in a given column

```
noNas = naexample[~isna(naexample[:,:y]),:]
```

Removing Rows with NA

naexample.csv

```
x,y
3,8
2,NA
,3
```

Return a new Dataframe with rows removed that have NA in a given column

```
noNas = naexample[~isna(naexample[:,:y]),:]
```

Condition

What to include
: means all columns

```
2×2 DataFrames.DataFrame
│ Row │ x  │ y │
├─────┼────┼───┤
│ 1   │ 3  │ 8 │
│ 2   │ NA │ 3 │
```

```
noNas = naexample[~isna(naexample[:,:y]),:x]
```

```
2-element DataArrays.DataArray{Int64,1}:
 3
  NA
```

Removing Rows with NA

naexample.csv

```
x,y
3,8
2,NA
,3
```

Delete rows in orginal Dataframe
that have NA in a given column

```
deleterows!(naexample,find(isna(naexample[:,:y])))
```

```
naexample
```

```
2×2 DataFrames.DataFrame
| Row | x  | y |
|-----|----|---|
| 1   | 3  | 8 |
| 2   | NA | 3 |
```

Subsets

```
large = DataFrame(A = 1:100,B = 1:2:200, C = rand(100))
large[1:3,[:C, :A]]
```

```
3×2 DataFrames.DataFrame
| Row | C        | A |
|-----|----------|---|
| 1   | 0.283288 | 1 |
| 2   | 0.645022 | 2 |
| 3   | 0.313295 | 3 |
```

Subset with condition

```
large = DataFrame(A = 1:100, C = rand(100))

large[large[:A] % 2 .== 0,:]
```

Only with two columns
Condition on first column

```
50×2 DataFrames.DataFrame
│ Row │ A   │ C        │
├─────┼─────┼──────────┤
│ 1   │ 2   │ 0.938738 │
│ 2   │ 4   │ 0.051108 │
│ 3   │ 6   │ 0.925504 │
│ 4   │ 8   │ 0.880809 │
│ 5   │ 10  │ 0.286541 │
│ 6   │ 12  │ 0.446456 │
│ 7   │ 14  │ 0.948686 │
│ 8   │ 16  │ 0.354929 │
⋮
│ 42  │ 84  │ 0.387136 │
│ 43  │ 86  │ 0.877899 │
│ 44  │ 88  │ 0.375194 │
```

Join

```
names = DataFrame(ID = [1, 2], Name = ["John Doe", "Jane Doe"])
jobs = DataFrame(ID = [1, 2], Job = ["Lawyer", "Doctor"])

full = join(names, jobs, on = :ID)
```

```
2×3 DataFrames.DataFrame
| Row | ID | Name       | Job      |
|-----|----|------------|----------|
| 1   | 1  | "John Doe" | "Lawyer" |
| 2   | 2  | "Jane Doe" | "Doctor" |
```

Types of Joins

Inner, Left

Right, Outer

Semi, Anti

Cross

Some Data for Examples

using RDatasets
iris = dataset("datasets", "iris")

150×5 DataFrames.DataFrame

Row	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	5.1	3.5	1.4	0.2	"setosa"
2	4.9	3.0	1.4	0.2	"setosa"
3	4.7	3.2	1.3	0.2	"setosa"
4	4.6	3.1	1.5	0.2	"setosa"
5	5.0	3.6	1.4	0.2	"setosa"
6	5.4	3.9	1.7	0.4	"setosa"
7	4.6	3.4	1.4	0.3	"setosa"
8	5.0	3.4	1.5	0.2	"setosa"
⋮					
142	6.9	3.1	5.1	2.3	"virginica"
143	5.8	2.7	5.1	1.9	"virginica"
144	6.8	3.2	5.9	2.3	"virginica"
145	6.7	3.3	5.7	2.5	"virginica"
146	6.7	3.0	5.2	2.3	"virginica"
147	6.3	2.5	5.0	1.9	"virginica"

Sepal is part of the plant that protects the flower in bud. IE the green part that covers the flower while it is forming. Some plants keep the sepal after the flower has bloomed.

Combining like values in a Column

```
by(iris, :Species, size)
```

```
3×2 DataFrames.DataFrame
| Row | Species      | x1      |
|-----|--------------|---------|
| 1   | "setosa"     | (50,5)  |
| 2   | "versicolor" | (50,5)  |
| 3   | "virginica"  | (50,5)  |
```

```
by(iris, :Species) do df
    DataFrame(N = size(df,1))
end
```

```
3×2 DataFrames.DataFrame
| Row | Species      | N  |
|-----|--------------|----|
| 1   | "setosa"     | 50 |
| 2   | "versicolor" | 50 |
| 3   | "virginica"  | 50 |
```


Thursday, September 15, 16

To rename a Column we can do:
a = by(iris, :Species, size)
rename!(a, “x1”, “N”)
Now the column in a is labeled “N”

Combining + Multiple Computed Values

```
by(iris, :Species) do df
    DataFrame(mean_petal_length = mean(df[:PetalLength]), s² = var(df[:PetalLength]))
end
```

```
3×3 DataFrames.DataFrame
| Row | Species      | mean_petal_length | s²         |
|-----|--------------|-------------------|------------|
| 1   | "setosa"     | 1.462             | 0.0301592  |
| 2   | "versicolor" | 4.26              | 0.220816   |
| 3   | "virginica"  | 5.552             | 0.304588   |
```

Combining - Apply function to each Column

aggregate(iris, :Species, sum)

```
3×5 DataFrames.DataFrame
| Row | Species      | SepalLength_sum | SepalWidth_sum | PetalLength_sum |
|-----|--------------|-----------------|----------------|-----------------|
| 1   | "setosa"     | 250.3           | 171.4          | 73.1            |
| 2   | "versicolor" | 296.8           | 138.5          | 213.0           |
| 3   | "virginica"  | 329.4           | 148.7          | 277.6           |

| Row | PetalWidth_sum |
|-----|----------------|
| 1   | 12.3           |
| 2   | 66.3           |
| 3   | 101.3          |
```

aggregate(iris, :Species, [sum, mean])

Sorting

Sorts by first column

Where first column is equal sort by second coloumn

sort!(iris)

150×6 DataFrames.DataFrame

Row	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	4.3	3.0	1.1	0.1	"setosa"
2	4.4	2.9	1.4	0.2	"setosa"
3	4.4	3.0	1.3	0.2	"setosa"
4	4.4	3.2	1.3	0.2	"setosa"
5	4.5	2.3	1.3	0.3	"setosa"
6	4.6	3.1	1.5	0.2	"setosa"
7	4.6	3.2	1.4	0.2	"setosa"
8	4.6	3.4	1.4	0.3	"setosa"
⋮					
142	7.2	3.6	6.1	2.5	"virginica"
143	7.3	2.9	6.3	1.8	"virginica"
144	7.4	2.8	6.1	1.9	"virginica"
145	7.6	3.0	6.6	2.1	"virginica"
146	7.7	2.6	6.9	2.3	"virginica"
147	7.7	2.8	6.7	2.0	"virginica"
148	7.7	3.0	6.1	2.3	"virginica"
149	7.7	3.8	6.7	2.2	"virginica"
150	7.9	3.8	6.4	2.0	"virginica"

DataFramesMeta - @with

```
using DataFrames, DataFramesMeta

sampledata = DataFrame(x = 1:3, y = [2, 1, 2])

  @with(sampledata, :y)                  # [2, 1, 2]

  @with(sampledata, :y + 1)              # [3, 2, 3]

  @with(sampledata, :y + :x)             # [3, 3, 5]

@with(sampledata, df[:x .> 1, ^(:y)])    # [1, 2]
```

^(:y) means pass y unchanged

@where - selecting rows

using DataFrames, DataFramesMeta

sampledata = DataFrame(x = 1:3, y = [2, 1, 2], z = [9, 4, 1])

@where(sampledata, :x .>1)

2×3 DataFrames.DataFrame

Row	x	y	z
1	2	1	4
2	3	2	1

@where(sampledata, :x .> 1, :y .< 2)

1×3 DataFrames.DataFrame

Row	x	y	z
1	2	1	4

@select - Selecting/Transforming Columns

using DataFrames, DataFramesMeta

sampledata = DataFrame(x = 1:3, y = [2, 1, 2], z = [9, 4, 1])

@where(sampledata, :x .>1)

2×3 DataFrames.DataFrame

Row	x	y	z
1	2	1	4
2	3	2	1

@where(sampledata, :x .> 1, :y .< 2)

1×3 DataFrames.DataFrame

Row	x	y	z
1	2	1	4

@transform - Adding Columns

using DataFrames, DataFramesMeta

sampledata = DataFrame(x = 1:3, y = [2, 1, 2], z = [9, 4, 1])

@where(sampledata, :x .>1)

2×3 DataFrames.DataFrame

Row	x	y	z
1	2	1	4
2	3	2	1

@where(sampledata, :x .> 1, :y .< 2)

1×3 DataFrames.DataFrame

Row	x	y	z
1	2	1	4