

# UTILITY FUNCTIONS IN R

DIYA DAS

GRADUATE STUDENT, NGAI LAB, DEPT OF MOLECULAR & CELL BIOLOGY

MOORE/SLOAN DATA SCIENCE FELLOW, BERKELEY INSTITUTE FOR DATA SCIENCE

# WHAT DO I MEAN BY UTILITY FUNCTIONS?

- Anything that makes using R easier as a \_\_\_\_\_
- For me, \_\_\_\_\_ is a molecular biologist analyzing RNA-sequencing data
- That's really it!
- Today's session: tips and tricks I've learned from 6.5 years of using R

# STARTING UP R: RUNNING THE SAME SCRIPTS

- When you get to your desk in the mornings, you probably putter around a bit to **set up your working environment** before you start your “actual work.”
- When you open R, you can do something similar. There are several (editable) setup files that are executed in the background.
- Because I’ve already edited mine, I’m going to use R -- vanilla to show you why you should edit your `.Rprofile`

# R “HACK” #0: R --vanilla

- Don't load any user profile (.Rprofile)
- Don't load any environment file (.Renviron)
- Don't load saved data file from last R session (.RData)

## Use cases:

1. When I've really messed up my configuration
2. When I'm not sure if I messed up my configuration but would like to be sure that my code is fully reproducible on other systems

# THE REASON I MADE AN .Rprofile

- I am a bit lazy and got annoyed with this prompt

```
> install.packages("nycflights13")
--- Please select a CRAN mirror for use in this session ---
Secure CRAN mirrors

1: 0-Cloud [https]                2: Algeria [https]
3: Australia (Canberra) [https]   4: Australia (Melbourne 1) [https]
5: Australia (Melbourne 2) [https] 6: Australia (Perth) [https]
7: Austria [https]               8: Belgium (Ghent) [https]
9: Brazil (PR) [https]           10: Brazil (RJ) [https]
11: Brazil (SP 1) [https]         12: Brazil (SP 2) [https]
13: Bulgaria [https]            14: Chile 1 [https]
15: Chile 2 [https]             16: China (Guangzhou) [https]
17: China (Lanzhou) [https]      18: Colombia (Cali) [https]
19: Czech Republic [https]      20: Denmark [https]
21: Ecuador (Cuenca) [https]    22: Estonia [https]
23: France (Lyon 1) [https]      24: France (Lyon 2) [https]
25: France (Marseille) [https]  26: France (Montpellier) [https]
27: France (Paris 2) [https]    28: Germany (Göttingen) [https]
29: Germany (Münster) [https]   30: Greece [https]
31: Iceland [https]            32: Indonesia (Jakarta) [https]
33: Ireland [https]            34: Italy (Padua) [https]
35: Japan (Tokyo) [https]       36: Malaysia [https]
37: Mexico (Mexico City) [https] 38: Norway [https]
39: Philippines [https]        40: Serbia [https]
41: Spain (A Coruña) [https]    42: Spain (Madrid) [https]
43: Sweden [https]             44: Switzerland [https]
45: Turkey (Denizli) [https]    46: Turkey (Mersin) [https]
47: UK (Bristol) [https]       48: UK (Cambridge) [https]
49: UK (London 1) [https]      50: USA (CA 1) [https]
51: USA (IA) [https]           52: USA (KS) [https]
53: USA (MI 1) [https]         54: USA (OR) [https]
55: USA (TN) [https]           56: USA (TX 1) [https]
57: Vietnam [https]           58: (other mirrors)
```

Selection:

# .Rprofile: CODE SNIPPETS/FUNCTIONS YOU'D LIKE TO HAVE AVAILABLE TO YOU ON STARTUP

- I install a lot of packages, so my .Rprofile is based around this:

```
options(repos=structure(c(CRAN="http://cran.cnr.berkeley.edu/",  
BioCsoft="http://www.bioconductor.org/packages/release/bioc/")))  
source("http://bioconductor.org/biocLite.R")  
  
ipakbio <- function(pkg){  
  new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]  
  if (length(new.pkg)) biocLite(new.pkg, dependencies = TRUE)  
  sapply(pkg, require, character.only = TRUE)  
} #based on https://gist.github.com/stevenworthington/3178163
```

Get this for yourself: <https://gist.github.com/diyadas>

# THE REASON I MADE AN .Renviron

(last week)

```
> suppressMessages(library(Seurat))
> suppressMessages(library(clusterExperiment))
> suppressMessages(library(scone))
Error: package or namespace load failed for 'scone'
in dyn.load(file, DLLpath = DLLpath, ...):
  unable to load shared object '/Library/Frameworks/R
.framework/Versions/3.4/Resources/library/RSQLite/li
bs/RSQLite.so':
  `maximal number of DLLs reached...
```

# .Renviron: ENVIRONMENT VARIABLES YOU'D LIKE TO SET ON STARTUP

- Not enough DLLs? Increase the max!

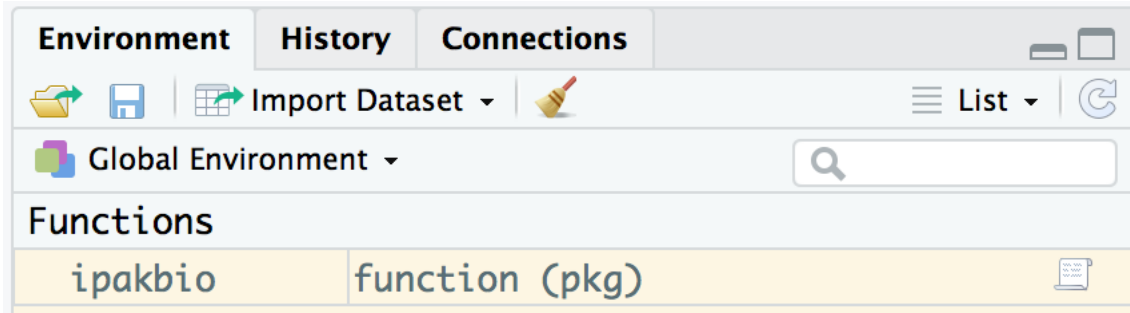
```
R_MAX_NUM_DLLS=150
```

- What is an environment variable?
  - Wikipedia: “An **environment variable** is a dynamic-named value that can affect the way running processes will behave on a computer.”
  - R: “It is impossible to list all the environment variables which can affect an **R** session...”



# (AB)USING ENVIRONMENTS IN R

- Start up RStudio; see the **Global Environment**



- What is an environment?
  - “A bag of names” that point to objects (i.e. dataframes, vectors), H. Wickham.
- So when you tell R to look for “a”, R looks in the environment for a name “a”

# HOW CAN YOU CHECK WHAT'S IN AN ENVIRONMENT?

- `ls()` lists all variables in the global environment

```
> ls()  
[1] "ipakbio"
```

- `rm(a)` removes “a” from the global environment, if it exists

```
> rm(a)  
Warning message:  
In rm(a) : object 'a' not found
```

- `rm(list=ls())` clears the global environment
  - Side note: CTRL + L is how to clear your console

# ENVIRONMENTS CAN BE NESTED

```
> f <- new.env()
> ls(f)
character(0)
> assign("a", 2, envir = f)
> ls()
[1] "f"          "ipakbio"
> ls(f)
[1] "a"
> f$a
[1] 2
> f$v <- 60
> ls(f)
[1] "a" "v"
> f$v
[1] 60
```

# ENVIRONMENTS CAN BE ATTACHED

- To attach an environment (or a dataframe, for that matter) is to add its variables to the search path
- You'll notice that `f` contains the variables `a`, `b`, and `v`, but calling only `b` from the global environment returns “not found”
- By adding `f` to the **search path**, we can avoid this issue

```
> ls(f)
[1] "a" "b" "v"
> b
Error: object 'b' not found
> attach(f)
> b
function(x) 3
```

Now that I've shown you how to do this, I'm also going to tell you why it's a horrible idea.

# THE SEARCH PATH TELLS R WHAT ENVIRONMENTS TO LOOK AT TO FIND VARIABLES (AND IN WHAT ORDER)

```
> attach(f)
> search()
[1] ".GlobalEnv"          "f"          "package:SummarizedExperiment"
[4] "package:DelayedArray" "package:matrixStats" "package:GenomicRanges"
[7] "package:GenomeInfoDb" "package:IRanges"    "package:S4Vectors"
[10] "package:stats4"       "package:Biobase"    "package:BiocGenerics"
[13] "package:parallel"    "tools:rstudio"      "package:stats"
[16] "package:graphics"    "package:grDevices"  "package:utils"
[19] "package:datasets"    "package:BiocInstaller" "package:methods"
[22] "Autoloads"           "package:base"

> detach(f)
> search()
[1] ".GlobalEnv"          "package:SummarizedExperiment" "package:DelayedArray"
[4] "package:matrixStats" "package:GenomicRanges"        "package:GenomeInfoDb"
[7] "package:IRanges"     "package:S4Vectors"            "package:stats4"
[10] "package:Biobase"     "package:BiocGenerics"         "package:parallel"
[13] "tools:rstudio"       "package:stats"                "package:graphics"
[16] "package:grDevices"   "package:utils"                "package:datasets"
[19] "package:BiocInstaller" "package:methods"              "Autoloads"
[22] "package:base"
```

# THIS CAN BE A BIT OF AN ISSUE WHEN LOADING...

- When you load everything into the global environment, objects with the same name will overwrite existing objects.
- See your startup messages when loading libraries.

```
> library(clusterExperiment)
Loading required package: SummarizedExperiment
Loading required package: GenomicRanges
Loading required package: stats4
Loading required package: BiocGenerics
Loading required package: parallel

Attaching package: 'BiocGenerics'

The following objects are masked from 'package:parallel':

  clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
  clusterExport, clusterMap, parApply, parCapply, parLapply,
  parLapplyLB, parRapply, parSapply, parSapplyLB

The following objects are masked from 'package:stats':

  IQR, mad, sd, var, xtabs

The following objects are masked from 'package:base':
```

(THIS ALSO LEADS TO FUN SITUATIONS)

```
> mean <- function(x) x+1  
> mean(c(4,60, 100))  
[1]    5   61 101  
> rm(mean)  
> mean(c(4,60, 100))  
[1] 54.66667
```

## A SLIGHT DETOUR: .RDATA FILES

- If you remember from earlier, I mentioned R --vanilla doesn't load .RData files in the current working directory
- These are “R Data” files that can save any R object – and then load them back as needed
- RStudio likes to save the contents of your workspace as an .RData file on exit
- But you can create them yourself with any collection of variables!

**All leading to the case where I created RData files named differently, but containing the same variable names and I wanted to compare them.**



# R.UTILS TO THE RESCUE!

- Henrik Bengtsson created a package I use for just one function: `loadToEnv()`
- You can load each Rda file into its own environment!
- Let's do some live coding.

# EXPORTED FUNCTIONS...

- In the live-coding demo, you may have seen me refer to `loadToEnv()` as `R.utils::loadToEnv()`
- This is a handy way of accessing just that function without loading the entire package
- In some cases, it's crucial to avoid overwriting functions from other packages. You can use also it to specify exactly which function in case you're not sure if it's been overwritten in the Global Env.
- The double colon notation allows you to find **exported functions** – functions you'd have been able to access **once you loaded the library**

**All good?**

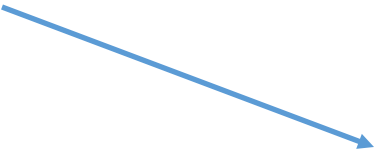
## ...AND NON-EXPORTED FUNCTIONS

- The functions you load when loading a package aren't everything in the package!
- They often have **helper functions** – functions that you won't have access to in the global environment but are often the “guts” of exported functions
- Good news: you can reach into the guts of a package and still access these non-exported functions

# INTRODUCING THE TRIPLE COLON OPERATOR

Write a function `pow(x, n)` that returns  $x^n$  where  $x$  is real and  $n$  is a positive number, without calling the built-in power function.

```
pow <- function(x, n){  
  return (prod(rep(x, n)))  
}
```



```
> pow(3,2)  
[1] 9  
> pow(2,3)  
[1] 8  
> pow(2.5,3)  
[1] 15.625  
> 2.5^3  
[1] 15.625  
> pow(2.5,3.5)  
[1] 15.625  
> 2.5^3.5  
[1] 24.70529
```

**ARGHHH**

# INTRODUCING THE TRIPLE COLON OPERATOR

Write a function `pow(x, n)` that returns  $x^n$  where  $x$  is real and  $n$  is a positive number, without calling the built-in power function.

```
pow <- function(x, n){  
  return (prod(rep(x, n)))  
}
```

The issue lies in using `rep` – you can't repeat anything a fractional number of times

But you can approximate  $n$  as a fraction...

And  $x^{a/b}$  is the same as  $(x^a)^{1/b}$

And  $y^{1/b}$  is the  $b$ th root of  $y$

So let's take the  $b$ th root of  $x^a$ !

# INTRODUCING THE TRIPLE COLON OPERATOR

Write a function `pow(x, n)` that returns  $x^n$  where  $x$  is real and  $n$  is a positive number, without calling the built-in power function.

```
> fractions(1.5)  
[1] 3/2
```

**There's this handy function in the MASS package to convert our exponent, but it doesn't return the numerator and the denominator separately**

# INTRODUCING THE TRIPLE COLON OPERATOR

Write a function `pow(x,n)` that returns  $x^n$  where  $x$  is real and  $n$  is a positive number, without calling the built-in power function.

```
> fractions(1.5)
[1] 3/2
```

**There's this handy function in the MASS package**

```
> fractions
function (x, cycles = 10, max.denominator = 2000, ...)
{
  ans <- .rat(x, cycles, max.denominator)
  ndc <- paste(ans$rat[, 1], ans$rat[, 2], sep = "/")
  int <- ans$rat[, 2] == 1
  ndc[int] <- as.character(ans$rat[int, 1])
  structure(ans$x, fracs = ndc, class = c("fractions", class(ans$x)))
}
<bytecode: 0x526935d48>
<environment: namespace:MASS>
```

# INTRODUCING THE TRIPLE COLON OPERATOR

Write a function `pow(x,n)` that returns  $x^n$  where  $x$  is real and  $n$  is a positive number, without calling the built-in power function.

**Ok, so we want whatever `“.rat”` is.**

```
> fractions
function (x, cycles = 10, max.denominator = 2000, ...)
{
  ans <- .rat(x, cycles, max.denominator)
  ndc <- paste(ans$rat[, 1], ans$rat[, 2], sep = "/")
  int <- ans$rat[, 2] == 1
  ndc[int] <- as.character(ans$rat[int, 1])
  structure(ans$x, fracs = ndc, class = c("fractions", class(ans$x)))
}
<bytecode: 0x526935d48>
<environment: namespace:MASS>
```



# INTRODUCING THE TRIPLE COLON OPERATOR

Write a function `pow(x,n)` that returns  $x^n$  where  $x$  is real and  $n$  is a positive number, without calling the built-in power function.

```
pow <- function(x, n){  
  if (is.integer(n)){  
    return (prod(rep(x, n)))  
  }  
  else {  
    myfrac <- MASS:::rat(n, 10, 2000)  
    a <- myfrac$rat[,1]  
    b <- myfrac$rat[,2]  
    return (pracma::nthroot(prod(rep(x, a)), b))  
  }  
}
```

