# Programming in R

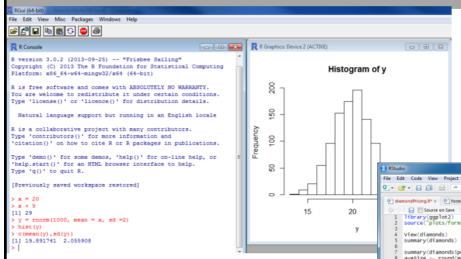
## What is R?

- Free software environment for statistical computing and graphics
- Available for Linux, Windows, MacOS
- For more information visit http://www.r-project.org/

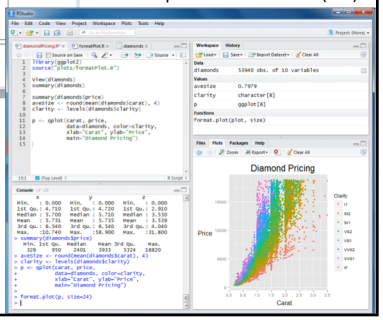
## What are we going to use R for today?

- RNAseq data QC (now)
- RNAseq differential expression analysis (now)
- Data visualization (after lunch)

## Ways to Use R - 1




R Graphical User Interface (GUI)




R Studio - Integrated Development Environment (IDE)

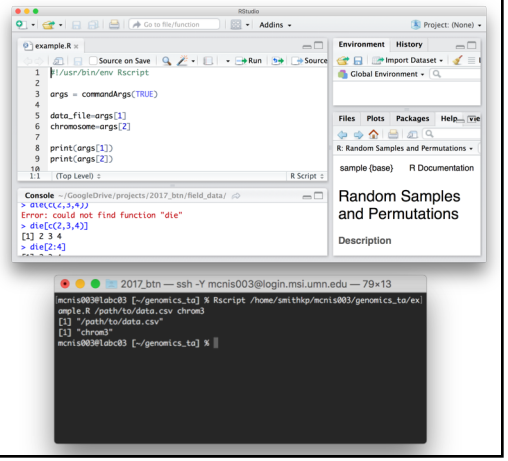
# Ways to Use R - 2

```
ag5431pi@labq02 [~] % module load R
ag5431pi@labq02 [~] % which R
/soft/R/3.1.1/bin/R
ag5431pi@labq02 [~] % R

R version 3.1.1 (2014-07-10) -- "Sock it to Me"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-unknown-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

R on the command line

# Ways to use R - 3

Entire R scripts can be run from the Bash command line using either R CMD BATCH or Rscript. Arguments can be added to the command that are used as variables in the script. Running an R script like this is useful because you can write a generic script for a common process and then provide new data and set variables easily through the arguments.




# Setting Up the R Environment

Determine your current working directory

> getwd()

NOTE: I will use
> to indicate an R command
# for comments
## for output

Set working directory to where you have your files

> setwd " /Users/cnhirsch/Desktop/example")

#Remember you can use tab completions!

Confirm you changed your current working directory with getwd()

List the contents of the directory

> list.files()

# Assigning variables

You can assign a value to a variable using " = " or " <- " , but the latter is more common

#Assign alex to the "name" variable
> name <- "alex"
> name

## "alex"

Tip: you can use underscores between words in variables, for example: field_data instead of fielddata.

Variables can be named anything you want, but there two rules

- Can't start with a number
- Can't contain the following characters: ^, !, $, @, +, -, /, *

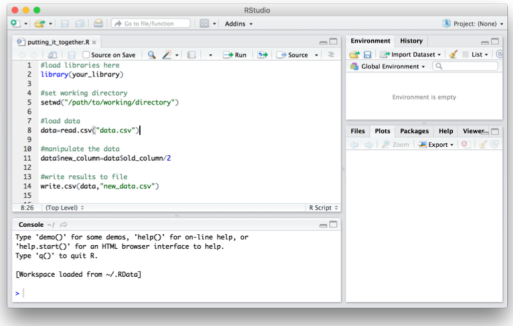
# Functions

R has a large number of built-in functions:

```
> round(3.1414)
## 3

> factorial(3)
## 6

> length(name)
## 1

> round(mean(1:6))
## 4
```

**Nested functions are performed from the inside out

# Putting it all together



Most R scripts follow this format to be easily understood by other researchers:

1. Libraries are load, the working directory is set, and data is imported first
2. Data is manipulated
3. Results are saved

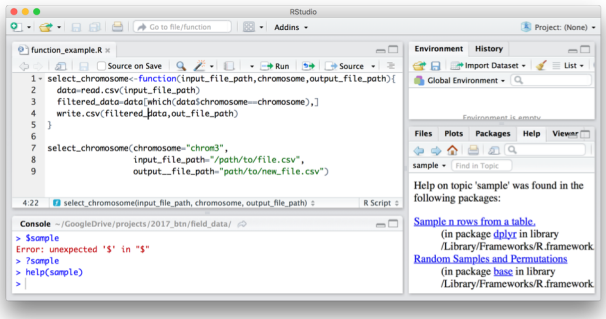
# Getting help

Don't know how to use a function or how it works?

```
# To see a detailed manual
> ?sample
> help(sample)

# To get basic usage info
> args(sample)

## function (x, size, replace = FALSE, prob = NULL)
```

Online help communities like Stack Overflow can also be useful and often provide more detail than official R documentation. For example, search for "r how to combine tables" to find help with combining tables.

# Example function

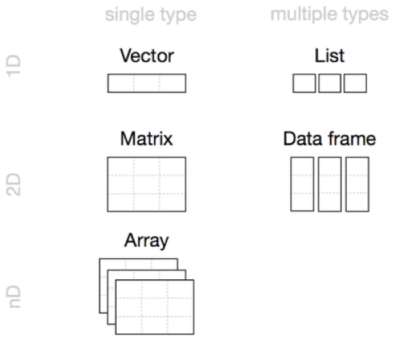


You can also define your own functions. User-defined functions simplify code making it easier to write, understand, and troubleshoot. For example, the user-defined function above will import a data file, filter the data file for a user defined chromosome, and write the results to a new file.

# Objects/Data Structures

Common data objects include:

- Vectors
- Lists
- Arrays
- Matrices
- Tables
- Data frames




# Atomic vectors

The atomic vector is the simplest R data type

Each atomic vector can only store one type of data:

- doubles
- integers
- characters
- logicals
- complex
- raw

#This is a double vector that stores regular numbers
> die <- c(1, 2, 3, 4, 5, 6)

#Don't just take my word for it…what if you test if it is a vector?
> is.vector(die)
*## TRUE*

> typeof(die)
*## "double"*

**Double is a computer science term that refers to to the specific number of bytes your computer used to store a number
-"numeric" is the more intuitive

# Working with vectors

Common operations:

```
> die <- c(1, 2, 3, 4, 5, 6)

#length()
> length(die)
## 6

#subset
> die[2:4]
##  2 3 4

#another way to subset
> die[c(2, 4, 6)]
## 2 4 6

#R is built to work with vectors
> die+1
## 2 3 4 5 6 7
```

# Let's see all this live now…

# Matrices

*Matrices* are used to store values in a two-dimensional array

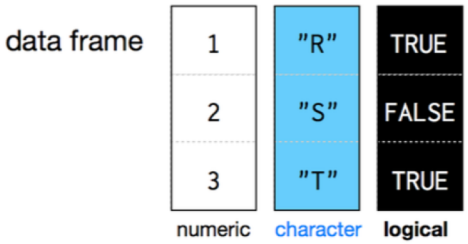
```
#Use our die vector to create a 2x3 matrix
> m <- matrix(die, nrow=2)
> m

##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
#how about a 3x2 matrix
> m <- matrix(die, ncol=2, byrow=TRUE)
> m

##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

# Data Frames

Most common way to store data

Essentially, a two-dimensional list




Data frames consist of column vectors and each column can be a different data type

# Data Frames

You can create a data frame by hand

```
> my_df <- data.frame(face = c("ace", "two", "six"),
      suit = c("clubs", "diamonds", "spades"), value = c(1, 2, 6))
> my_df

##   face suit  value
##   ace  clubs     1
##   two  diamonds  2
##   six  spades    6
```

But...

It's often easier to create one by loading in data

# Loading data

Numerous functions to load in specific types of data

read.table() will work for most data types, read.csv() is a wrapper of read.table() with defaults set for csv format

**Our example data**

Mammal life-history, geography, and ecology traits from the PanTHERIA database:

> Jones, K.E., et al. PanTHERIA: a species-level database of life history, ecology, and geography of extant and recently extinct mammals. Ecology 90:2648. http://esapubs.org/archive/ecol/E090/184/

## Getting started

```
# Use read.table to load-in the data
> mammals <- read.table("Lab2-R-Part1/mammals_dataset.txt", sep="\t", stringsAsFactors=FALSE)

# Look at structure of data
> str(mammals)

# Get a peak of the data
> head(mammals)
```

## In Class Exercise

1. What are the dimensions of this dataset?
2. Look at the last few lines of the dataset
3. Are there any missing values in the "adult_body_mass_g" column?
4. Find the mean value of the "adult_body_mass_g" column
5. How many times does each order appear in the dataset?

## In Class Practice - Solutions

1. What are the dimensions of this dataset?
   - dim(mammals)
2. Look at the last the few lines of the dataset
   - tail(mammals)
3. Are there any missing values in the "adult_body_mass_g" column?
   - sum(is.na(mammals$adult_body_mass_g))
4. Find the mean value of the "adult_body_mass_g" column?
   - mean(mammals$adult_body_mass_g, na.rm=TRUE)
5. How many times does each order appear in the dataset?
   - table(mammals$order)

## R Takeaways

- Powerful tool for statistical analysis and data visualization
- Able to interface with R in many different ways (GUI, Rstudio, command line)
- Many built in functions (you can also make your own functions!)
- Like many things in bioinformatics, there is a learning curve, but learning R is time well spent