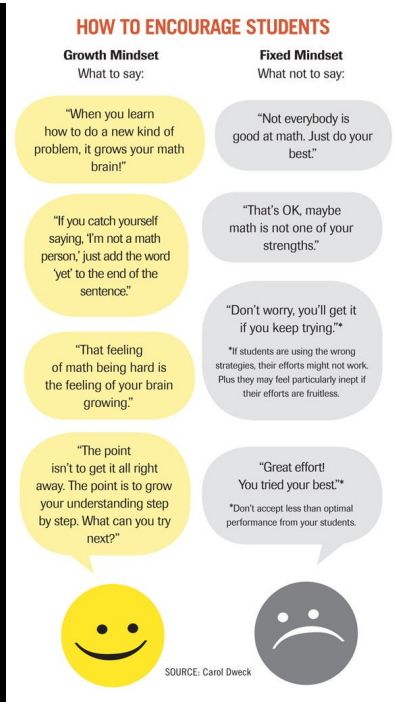


Side-Note:

Replace "**math**" with "**R programing**" to help reframe your self criticism when learning…

Recap From Last Time:

- Why it is important to visualize data during exploratory data analysis.
- Discussed data visualization best practices and how good visualizations optimize for the human visual system.
- Introduced the extensive graphical capabilities of base R with a focus on generating and customizing scatterplots, histograms, bar graphs, boxplots, (dendrograms and heatmaps).
- Use the par() function to control fine grained details of the afore mentioned plot types.
- Used (and SAVED!) an R script with all our work!

[MPA Link]

Today's Learning Goals

- **Last days R visualization hands-on exercise revisited…**
  - Make our work error free and produce a nice report!
- **More on data import**
  - File pre-check recommendations
  - Using read.table() and friends for flat files
- **Writing your own functions**
  - What, Why, When and How
- **Hands-on session**
  - Practice, tips, techniques for troubleshooting, and best practice guidelines for writing and debugging your functions

Today's Learning Goals

- **Last days R visualization hands-on exercise revisited…**
  - Make our work error free and produce a nice report!
- **More on data import**
  - File pre-check recommendations
  - Using read.table() and friends for flat files
- **Writing your own functions**
  - What, Why, When and How
- **Hands-on session**
  - Practice, tips, techniques for troubleshooting, and best practice guidelines for writing and debugging your functions

Class 5 Revisited

- Open your previous class05 RStudio **project** (and your saved **R script**)
- Can you source this class05.R file to re-generate all your plots without error?

- If so you can now generate a nice HTML report of your work to date...

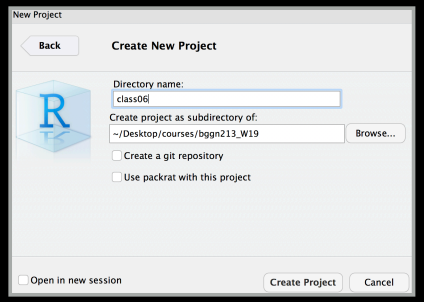
[Take 2-3 minutes]

Today's Learning Goals

- **Last days R visualization hands-on exercise revisited…**
  - Make our work error free and produce a nice report!
- **More on data import**
  - File pre-check recommendations
  - Using read.table() and friends for flat files
- **Writing your own functions**
  - What, Why, When and How
- **Hands-on session**
  - Practice, tips, techniques for troubleshooting, and best practice guidelines for writing and debugging your functions

Pre-check recommendations

- **Get organized!**
  - Start a new 'project' in your class directory (e.g. a folder on your Desktop where you are storing all your class material)
  - In RStudio **File > New Project > New Directory >** … class06



Pre-check recommendations

- **Get organized!**
  - Start a new 'project' in your class directory (e.g. a folder on your Desktop where you are storing all your class material)
  - In RStudio **File > New Project > New Directory >** … class06
- **Inspect the file**
  - Move the input file into your project directory
  - Open it from the RStudio **Files** tab.
  - Does it have a header line or comments to be included, ignored or removed?
  - Avoid file (or field names) with spaces or special characters such as ?, $,%, ^, &, *, } etc.
  - Short names are preferred over longer names.
  - Does the file end with a blank line or a RTN?

read.table() and friends for flat files

- **The read.table() function is the base of all flat file import functions**
  - **read.delim**("filename.txt", sep="\t") TAB
  - **read.csv**("filename.txt", sep=",") COMMA
  - **read.csv2**("filename.txt", sep=";") SEMI-COLON
  - **read.table**("filename.txt", sep="") WHITE SPACE
- **What other differences are there between these functions?**
- **MS EXCEL file import options include:**
  - Export (i.e. "Save As…") your excel data to plain text **CSV format**.
  - Or if you must, use **readxl::read_excel()** to read specified parts of your sheets.
- **For fast and convenient reading of very large flat files files**
  - Try **data.table::fread()** use is similar to **read.table()** but it automatically finds field separators and header rows. It is also much faster!
- **Saving and loading .RData files…**
  - Use the functions **save()** and **load()** for saving and loading multiple objects to space efficient binary format files.

Your turn!

Do it Yourself!

https://bioboot.github.io/bggn213_S18/class-material/test1.txt
https://bioboot.github.io/bggn213_S18/class-material/test2.txt
https://bioboot.github.io/bggn213_S18/class-material/test3.txt

- Start a new RStudio **Project** in a clean directory
- Open a new Rmarkdown document and give it a name and descriptive text.
- Download each of the above files and move them into your *Project*
- Experiment with **read.table()** to get their data successfully input into your R session.

Today's Learning Goals

- **Last days R visualization hands-on exercise revisited…**
  - Make our work error free and produce a nice report!
- **More on data import**
  - File pre-check recommendations
  - Using read.table() and friends for flat files
- **Writing your own functions**
  - What, Why, When and How
- **Hands-on session**
  - Practice, tips, techniques for troubleshooting, and best practice guidelines for writing and debugging your functions

What is a function

```
name.of.function <- function(arg1, arg2) {
    statements
    return(something)
}
```

1. **Name** (can be *almost* anything you want)
2. **Arguments** (i.e. input to your function)
3. **Body** (where the work gets done)

What is a function

```
add <- function(x, y=1) {
    # Sum the input x and y
    x + y
}
```

1. **Name** (in this case "**add**")
2. **Arguments** (here "**x**" and "**y**")
3. **Body** (will return the result of the last statement)

Your function is treated just like any other function…

```
add <- function(x, y=1) {
    # Sum the input x and y
    x + y
}

add(x=1, y=4)
add(1, 4)
add(1)

add( c(1, 2, 3) )
add( c(1, 2, 3), 4 )

add(1, 2, 2)
add(x=1, y="b")
```

Why would you write a function

When you find yourself doing the same thing 3 or more times it is time to write a function.

```
## What does this code do?

df$a <- (df$a - min(df$a)) / (max(df$a) - min(df$a))

df$b <- (df$b - min(df$a)) / (max(df$b) - min(df$b))

df$c <- (df$c - min(df$c)) / (max(df$c) - min(df$c))

df$d <- (df$d - min(df$d)) / (max(df$a) - min(df$d))
```

Why would you write a function

When you find yourself doing the same thing 3 or more times it is time to write a function.

```
## Consider copy and paste errors:

df$a <- (df$a - min(df$a)) / (max(df$a) - min(df$a))

df$b <- (df$b - min(df$a)) / (max(df$b) - min(df$b))

df$c <- (df$c - min(df$c)) / (max(df$c) - min(df$c))

df$d <- (df$d - min(df$d)) / (max(df$a) - min(df$d))
```

Why would you write a function

Consider the advantages:

```
## Here the intent is far more clear

df$a <- rescale(df$a)
```

- Makes the purpose of the code more clear
- Reduce mistakes from copy/paste
- Makes updating your code easer
- Reduce duplication and facilitate re-use.

How would you write this function

Start with a **working code snippet**, simplify, reduce calculation duplication,…

```
## First consider the original code:

df$a <- (df$a - min(df$a)) / (max(df$a) - min(df$a))

df$b <- (df$b - min(df$a)) / (max(df$b) - min(df$b))

df$c <- (df$c - min(df$c)) / (max(df$c) - min(df$c))

df$d <- (df$d - min(df$d)) / (max(df$a) - min(df$d))
```

How would you write this function

Start with a working code snippet, **simplify**, reduce calculation duplication,…

```
## Simplify to work with a generic vector named "x"

x <- (x - min(x)) / (max(x) - min(x))
```

How would you write this function

Start with a working code snippet, simplify, **reduce calculation duplication**,…

```
## Note that we call the min() function twice…

x <- (x - min(x)) / (max(x) - min(x))
```

How would you write this function

Start with a working code snippet, simplify, **reduce calculation duplication**,…

```
## Note that we call the min() function twice…

xmin <- min(x)
x <- (x - xmin) / (max(x) - xmin)
```

How would you write this function

Start with a working code snippet, simplify, **reduce calculation duplication**,…

```
## Further optimization to use the range() function…

rng <- range(x)
x <- (x - rng[1]) / (rng[2] - rng[1])
```

How would you write this function

Start with a working code snippet, simplify, reduce calculation duplication, finally turn it into a **function**

```
## You need a “name”, “arguments” and “body”…

rescale <- function(x) {
   rng <-range(x)
   (x - rng[1]) / (rng[2] - rng[1])
}

# Test on a small example where you know the answer
rescale(1:10)
```

How would you write this function

Test, **Fail**, Change, Test again,…

```
# Test on a small example where you know the answer
rescale(1:10)

# How would you get your function to work here…
rescale( c(1,2,NA,3,10) )

# What should your function do here?
recale( c(1,10,"string") )
```

Side-Note: Seeing and using your function in RStudio

- An easy way to visualize the code of a function is to type its name without the parentheses ().
- If you have your new function saved to a separate file then you can load and execute it using the **source()** function. E.g. **source("MyUtils.R")**
- The **return()** statement is not required in a function but it is advisable to use it when the function performs several computations. It has the effect of ending the function execution and returning control to the code which called it.

```
rescale <- function(x, na.rm=TRUE, plot=FALSE) {
   if(na.rm) {
     rng <-range(x, na.rm=na.rm)
   } else {
     rng <-range(x)
   }
   print("Hello")

   answer <- (x - rng[1]) / (rng[2] - rng[1])

   print("is it me you are looking for?")

   if(plot) {
      plot(answer, typ="b", lwd=4)
   }
   print("I can see it in ...")
}
```

```
rescale <- function(x, na.rm=TRUE, plot=FALSE) {
   if(na.rm) {
     rng <-range(x, na.rm=TRUE)
   } else {
     rng <-range(x)
   }
   print("Hello")

   answer <- (x - rng[1]) / (rng[2] - rng[1])
   return(answer)
   print("is it me you are looking for?")

   if(plot) {
      plot(answer, typ="b", lwd=4)
   }
   print("I can see it in ...")
}
```

Today's Learning Goals

- **Last days R visualization hands-on exercise revisited…**
  - Make our work error free and produce a nice report!
- **More on data import**
  - File pre-check recommendations
  - Using read.table() and friends for flat files
- **Writing your own functions**
  - What, Why, When and How
- **Hands-on session**
  - Practice, tips, techniques for troubleshooting, and best practice guidelines for writing and debugging your functions

Do it Yourself!

Your turn!

https://bioboot.github.io/bggn213_W19/lectures/#6

Concentrate on **Section 1B** and questions 1 to 6.
Other sections are there for your benefit.

[Also aim to generate a HTML report from your R script]

Do it Yourself!

```
# Can you improve this analysis code?
library(bio3d)
s1 <- read.pdb("4AKE")  # kinase with drug
s2 <- read.pdb("1AKE")  # kinase no drug
s3 <- read.pdb("1E4Y")  # kinase with drug

s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")
s3.chainA <- trim.pdb(s1, chain="A", elety="CA")

s1.b <- s1.chainA$atom$b
s2.b <- s2.chainA$atom$b
s3.b <- s3.chainA$atom$b

plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
plotb3(s2.b, sse=s2.chainA, typ="l", ylab="Bfactor")
plotb3(s3.b, sse=s3.chainA, typ="l", ylab="Bfactor")
```

Homework!

New **DataCamp** Assignments

- Introduction to R Markdown
- Functions
- Loops

Muddy Point Assessment Form Link

Side-Note:

Replace "**math**" with "**R programing**" to help reframe your self criticism when learning…