HOW TO ENCOURAGE STUDENTS

Growth Mindset	Fixed Mindset
What to say:	What not to say:
"When you learn how to do a new kind of problem, it grows your math brain!"	"Not everybody is good at math. Just do your best."
"If you catch yourself saying, 'I'm not a math person,' just add the word 'yet' to the end of the sentence."	"That's OK, maybe math is not one of your strengths."
"That feeling of math being hard is the feeling of your brain growing."	"Don't worry, you'll get it if you keep trying."*
	*If students are using the wrong strategies, their efforts might not work. Plus they may feel particularly inept if their efforts are fruitless.
"The point isn't to get it all right away. The point is to grow your understanding step by step. What can you try next?"	"Great effort! You tried your best."*
	*Don't accept less than optimal performance from your students.

SOURCE: Carol Dweck

Side-Note:

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

BGGN 213
R Functions
Lecture 6

Barry Grant
UC San Diego

http://thegrantlab.org/bggn213

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.

- Stoped hands-on session at section 2C

[MPA Link]

Today's Learning Goals

- **Last days R visualization hands-on exercise 2C revisited…**
  - Understanding, using and improving someone else function

- **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 2C revisited...**
  - Understanding, using and improving someone else function

- **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?

  ① ②
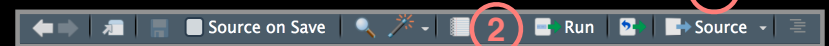
- If so you can now generate a nice HTML report of your work to date...

  [Take 2-3 minutes]

Exercise 4 Revisited

- Locate and open in RStudio the downloaded file color_to_value_map.r

- This is an example of a poorly written function typical of something you might get from a lab mate that knows some R...

  [Take 2-3 minutes]

(POOR!) Color Mapping Function

```
map.colors <- function (value,high.low,palette) {
  proportion <- ((value-high.low[1])/(high.low[2]-high.low[1]))
  index <- round ((length(palette)-1)*proportion)+1
  return (palette[index])
}
```

Talking points:
- Can you figure out what this function it is supposed to do?
- What format should the inputs be in order to work?
- How could we improve this function?

1. What are the function inputs?

```
map.colors2 <- function(x, high.low, palette) {

  proportion <- ((x - high.low[1])/(high.low[2] - high.low[1]))

  index <- round( (length(palette)-1) * proportion )+1

  return(palette[index])
}
```

Let's first space things out so it is easier for us to read and then change to use **x** as our numeric input vector.

1. What are the function inputs?

```
map.colors2 <- function(x, high.low, palette) {

  proportion <- ((x - high.low[1])/(high.low[2] - high.low[1]))

  index <- round( (length(palette)-1) * proportion )+1

  return(palette[index])
}
```

Let's first space things out so it is easier for us to read and then change to use **x** as our numeric input vector.

We can guess that `high.low` is a two element numeric vector and `palette` is probably a vector of colors

2. What is the function doing?

```
map.colors2 <- function(x, high.low, palette) {

  # Determine precent values of the 'high.low' range
  proportion <- ((x - high.low[1])/(high.low[2] - high.low[1]))

  index <- round( (length(palette)-1) * proportion )+1

  return(palette[index])
}
```

Let's add a **comment** to explain the logic of the first line

2. What is the function doing?

```
map.colors2 <- function(x, high.low, palette) {

  # Determine precent values of the 'high.low' range
  precent <- ((x - high.low[1])/(high.low[2] - high.low[1]))

  index <- round( (length(palette)-1) * precent )+1

  return(palette[index])
}
```

Let's change the object name from `proportion` to **precent** so it is more meaningful for us. Remember to change it everywhere ;-)

2. What is the function doing?

```
map.colors2 <- function(x, high.low, palette) {

  # Determine precent values of the 'high.low' range
  precent <- ((x - high.low[1])/(high.low[2] - high.low[1]))

  #index <- round( (length(palette)-1) * precent )+1
  index <- round( length(palette) * precent )

  return(palette[index])
}
```

Perhaps we can simplify the next line, which determines the corresponding index position in the color 'palette' vector?

2. What is the function doing?

```
map.colors2 <- function(x, high.low, palette) {

  # Determine precent values of the 'high.low' range
  precent <- ((x - high.low[1])/(high.low[2] - high.low[1]))

  #index <- round( (length(palette)-1) * precent )+1
  index <- round( length(palette) * precent )

  return(palette[index])
}
```

Doh! What happens if our `precent` value is zero or very small?

We will get an **index** value of zero, will cause a problem when accessing palette[**index**] in the last line

2. What is the function doing?

```
map.colors2 <- function(x, high.low, palette) {

  # Determine precent values of the 'high.low' range
  precent <- ((x - high.low[1])/(high.low[2] - high.low[1]))

  # Find corresponding index position in the color 'palette'
  #   note catch for 0 precent values to 1
  index <- round( (length(palette)-1) * precent )+1

  return(palette[index])
}
```

Add a comment again to describe the logic of what our code is doing

3. How could we improve this function?

```
map.colors2 <- function(x, high.low, palette) {

  ## Description: Map the values of the input vector 'x'
  ##   to the input colors vector 'palette'

  # Determine precent values of the 'high.low' range
  precent <- ((x - high.low[1])/(high.low[2] - high.low[1]))

  # Find corresponding index position in the color 'palette'
  #   note catch for 0 precent values to 1
  index <- round( (length(palette)-1) * precent )+1

  return(palette[index])
}
```

Make more user friendly in lots of ways including adding **description**, input argument defaults, error checking of inputs, tests, etc.

3. How could we improve this function?

```
map.colors3 <- function(x,
                        low.high = range(x),
                        palette  = cm.colors(100)) {

  ## Description: Map the values of the input vector 'x'
  ##   to the input colors vector 'palette'

  # Determine precent values of the 'high.low' range
  precent <- ((x - low.high[2])/(low.high[1] - low.high[2]))

  # Find corresponding index position in the color 'palette'
  #   note catch for 0 precent values to 1
  index <- round( (length(palette)-1) * precent )+1

  return(palette[index])
}
```

Make more user friendly in lots of ways including adding description, input argument defaults, error checking of inputs, tests etc.

Today's Learning Goals

· **Last days R visualization hands-on exercise 2C revisited…**
  • Understanding, using and improving someone else function

· **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 a directory you know about and store all needed project material here (input, scripts and output).
  • In RStudio **File > New Project > New Directory >** …

Pre-check recommendations

• **Get organized!**
  • Start a new '**project**' in a directory you know about and store all needed project material there (input, scripts and output).
    • In RStudio **File > New Project > New Directory >** …

• **Inspect the file**
  • 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.

Do it Yourself!

Your turn!

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 2C revisited…**
  - Understanding, using and improving someone else function

- **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

```
                1                          2
name.of.function <- function(arg1, arg2) {
    statements              3
    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

```
1                    2
add <- function(x, y=1) {
    # Sum the input x and y      3
    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.

```r
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 ...")
}
```

```r
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 2C revisited…**
  - Understanding, using and improving someone else function

- **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

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]

```
# 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

HOW TO ENCOURAGE STUDENTS

Growth Mindset	Fixed Mindset
What to say:	What not to say:

"When you learn how to do a new kind of problem, it grows your math brain!"

"Not everybody is good at math. Just do your best."

"If you catch yourself saying, 'I'm not a math person,' just add the word 'yet' to the end of the sentence."

"That's OK, maybe math is not one of your strengths."

"Don't worry, you'll get it if you keep trying."*

*If students are using the wrong strategies, their efforts might not work. Plus they may feel particularly inept if their efforts are fruitless.

"That feeling of math being hard is the feeling of your brain growing."

"The point isn't to get it all right away. The point is to grow your understanding step by step. What can you try next?"

"Great effort! You tried your best."*

*Don't accept less than optimal performance from your students.

SOURCE: Carol Dweck

Side-Note:

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