

# Grading Code with gradethis

NYR Conference 2020

Daniel Chen @chendaniely

Virginia Tech

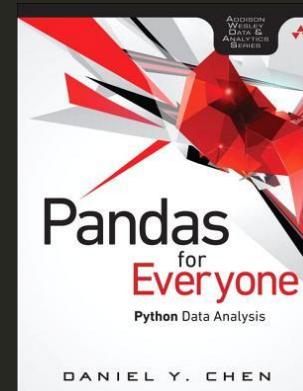
2020-08-11

1  
h  
ai

# I'm Daniel



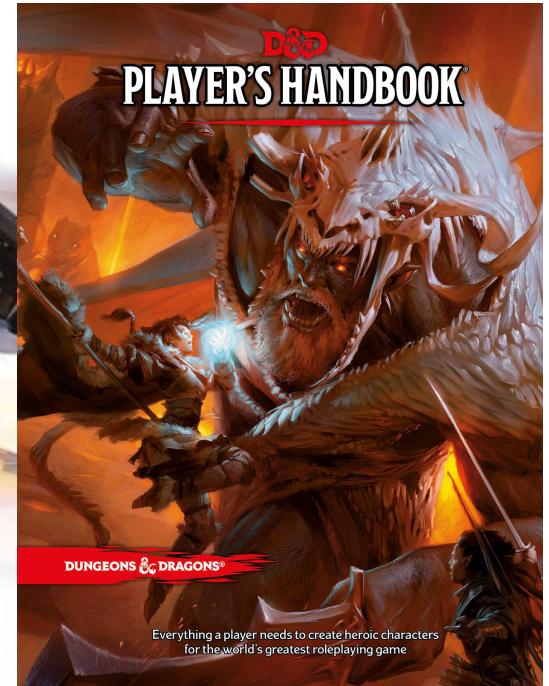
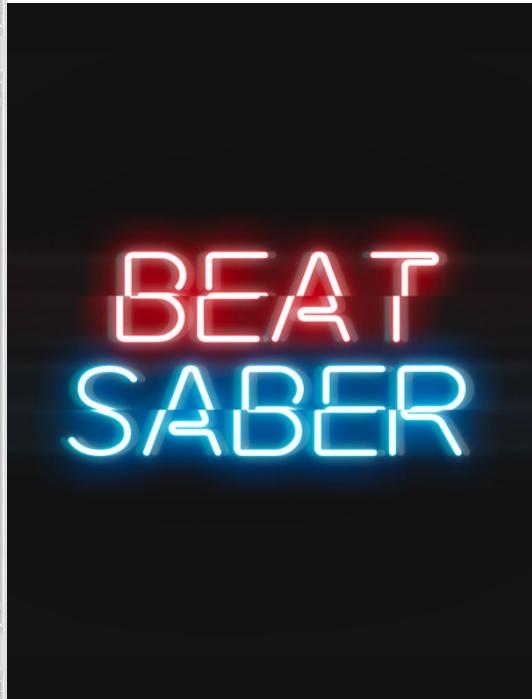
- PhD Student: Virginia Tech
  - Data Science education & pedagogy
  - Medical, Biomedical, Health Sciences
  - <https://github.com/chendaniely/dissertation>
  - Advisor: Anne Brown, PhD
    - <https://www.databridge.dev/>
    - <https://bevanbrownlab.com/>
- 2019 Intern at RStudio
  - **gradethis**
  - Code grader for **learnr** documents
- Author:



# How's Quarantine?



# How's Quarantine?



[https://github.com/chendaniely/animal\\_crossing](https://github.com/chendaniely/animal_crossing)

<https://github.com/chendaniely/animalcrossing>

<https://www.twitch.tv/chendaniely>

# Hobbes -- Blue Heeler



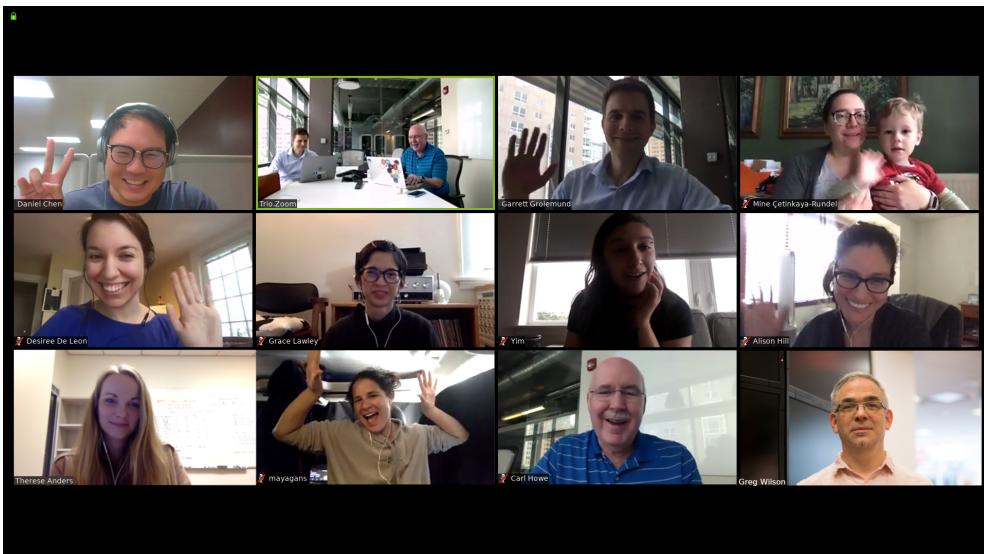
# RStudio Internship

<https://rstudio-education.github.io/gradethis/>

- Worked on `gradethis` for my RStudio 2019 Internship
  - Barret Schloerke, PhD
  - Garrett Grolemund, PhD
- I am a user of R, not a developer.
- Wrote about my experiences:
  - My time as an intern
  - Internship week 1
  - Internship week 2
  - Moved to blogdown
  - Git workflow
  - Rstudio education blog post

# Education + Shiny Team

Education Team



Shiny Team



# Join all the slack channels!

- What? A book about shiny?
- Yes! <https://mastering-shiny.org/>
- No ):

Tuesday, June 18th

 **Daniel Chen** 10:19 PM  
joined #shiny-book.

Thursday, June 20th

 **hadley** 9:45 AM  
@Daniel Chen this is a place for me and Joe to chat about the shiny-book. I don't think it's something you should be following at this point

 **Daniel Chen** 9:56 AM  
left #shiny-book.

:D



Hadley Wickham ✅  
@hadleywickham

Thanks to Daniel Chen for the awesome #rstats bow tie!



Daniel Chen 🐍 💀  
@chendaniely

Omg. Is @hadleywickham wearing what I think he's  
wearing???? 😂,:,😭😭😭😭

#rstudioconf2020

# Learnr interactive tutorials

<https://rstudio.github.io/learnr/>

1. Narrative, figures, illustrations, and equations.
2. Code exercises (R code chunks that users can edit and execute directly).
3. Quiz questions.
4. Videos (supported services include YouTube and Vimeo).
5. Interactive Shiny components.

# Creating a `learnr` tutorial

```
---
```

```
title: "Hello, Tutorial!"
```

```
output: learnr::tutorial
```

```
runtime: shiny_prerendered
```

```
--
```

```
```{r setup, include=FALSE}
```

```
library(learnr)
```

```
```
```

This code computes the answer to one plus one, change it so it computes two plus two:

```
```{r addition, exercise=TRUE}
```

```
1 + 1
```

```
```
```

# Learnr output

```
1 ---  
2 title: "Hello, Tutorial!"  
3 output: learnr::tutorial  
4 runtime: shiny_prerendered  
5 ---  
6  
7 ```{r setup, include=FALSE}  
8 library(learnr)  
9 ```  
10  
11 This code computes the answer to one plus one, change it so it computes two plus two:  
12  
13 ```{r addition, exercise=TRUE}  
14 1 + 1  
15 ...
```

This is what the running tutorial document looks like after the user has entered their answer:

Hello, Tutorial!

J.J. Allaire  
March 1st, 2017

Start Over

The following code computes the answer to  $1+1$ . Change it so it computes  $2 + 2$ :

Code

1 1 + 1  
2  
3

[Run Code]

[1] 2

# Sortable

- More `learnr` examples: <https://rstudio.github.io/learnr/examples.html>
  - `sortable`: [https://andrie-de-vries.shinyapps.io/sortable\\_tutorial\\_question\\_rank/](https://andrie-de-vries.shinyapps.io/sortable_tutorial_question_rank/)

**Exercise**

Drag the items in any desired order

one

two

three

*Complex html tag without a name*

*Complex html tag with name: 'five'*

**Result**

```
[1] "one"                                "two"  
[3] "three"                             "Complex html tag without a name"  
[5] "five"
```

# Sortable -> Parsons

- Parson problems are coding exercises designed to **reduce cognitive load** on the learner.
  - Instead of writing the code, they order code blocks in the correct order
  - Reduces syntax errors
  - Learn the "what" concepts instead of the "how" syntax

<https://github.com/rstudio/parsons>

# Parsons

Drag from here

iris

print()

summarize(...)

mutate(...)

Construct your solution here

Give feedback

# Grading the questions

We can create questions, but how do we give feedback?

At the very least how to we tell the student the solution they provided was "wrong"?

# Look at the result

- Only check if the solutions match
- As long as the correct answer is returned, it doesn't matter how the student got the answer
  - Can't check if you are trying to teach a specific function

E.g.,  $3 + 3 = 6 = 2 * 3$

# Look at the code

- Essentially compares the actual code (e.g., the text)
- It's very strict
  - `3 + 3 != 2 * 3`
  - `apply(df, 2, class) != apply(df, margin = 2, FUN = class)`
  - `mean(1:5) != mean(1:5, na.rm=FALSE)`

# Look at the code's AST

- The Abstract Syntax Tree (AST) is a graphical (i.e., tree) representation of your code.
- It shows the order of function calls and their arguments to be executed.

```
lobstr::ast(  
  log(exp(3))  
)
```

```
## o-log  
## \-o-exp  
##   \-3
```

```
lobstr::ast(  
  log(exp(2))  
)
```

```
## o-log  
## \-o-exp  
##   \-2
```

```
lobstr::ast(  
  log(log(2, base = 10))  
)
```

```
## o-log  
## \-o-log  
##   +-2  
##     \base = 10
```

# Gradethis

<https://rstudio-education.github.io/gradethis/>

- Auto grading system that can be used to grade `learnr` exercises.
  - Check the answer
  - Check the code (using the AST)
- In `learnr`, each exercise is completely independent from the others
- What makes it unique is its ability to give formative feedback to the student.
- `leanr` but why not `grader`?
  - The package was originally named `grader`
  - Someone from UVA took the name on CRAN during the first few weeks of the internship.
  - <https://github.com/rstudio-education/gradethis/issues/18>

# Formative feedback

Here is a number. You can do great things with it, like this:

- Take the square root of the log of the number 2.

Then click Submit Answer.

Code

 Start Over  Hints  Run Code  Submit Answer

|   |   |
|---|---|
| 1 | 2 |
| 2 |   |
| 3 |   |

```
[1] 2
```

I expected a call to `sqrt()` where you wrote 2. Try it again; next time's the charm!

# learnr + gradethis

1. base chunk: what the student sees in the exercise
2. -check chunk: how should the student's code be graded
  1. solution: are you just checking the final result?
    - `gradethis::grade_result()`
    - Provide a set of conditions to check, returns result on *first match* (i.e., order of conditions matter)
  2. code: are you checking to see the code itself is correct (AST)
    - `gradethis::grade_code()`
    - -solution chunk: The instructor's solution code that will be used to compare to the student's code
  3. unittests: are you testing a function that needs to pass a bunch of checks?
    - `gradethis::grade_conditions()`
3. -hint chunks (optional)

# learnr

Set the checking function for `learnr`

```
```{r setup}
library(gradethis)
tutorial_options(exercise.checker = gradethis::grade_learnr)
```
```

We can scaffold the exercise to reduce cognitive load.

```
```{r chunkLabel, exercise=TRUE}
log(____(____))
```
```

And want the student to enter

```
```{r chunkLabel, exercise=TRUE}
log(exp(3))
```
```

# learnr + gradethis::grade\_result()

```
```{r chunkLabel, exercise=TRUE}
log(____(____))
```

```{r chunkLabel-hint-1}
# hint text
"Exponentiate with the `exp` function."
```

```{r chunkLabel-check}
# check result
gradethis::grade_result(
  # pass_if fail_if order does matter. Maybe put pass_if conditions last?

  #.result is the last value returned by the student
  gradethis::pass_if(~ identical(.result, 3), "YAY!"),
  gradethis::fail_if(function(x){identical(x, 4)}, "4 is an incorrect input.")
  gradethis::fail_if(2, "2 was the wrong number to use here.")
)
```
``
```

# learnr + gradethis::grade\_code()

```
```{r chunkLabel, exercise=TRUE}
log(____(____))
```

```{r chunkLabel-hint-1}
# hint text
"Exponentiate with the `exp` function."
```

```{r chunkLabel-solution}
# solution code
log(exp(3))
```

```{r chunkLabel-check}
# check code
gradethis::grade_code()
```
```

# learnr + gradethis::grade\_conditions()

```
```{r grade_conditions, exercise = TRUE}
# student code
add_1 <- function(x) {x + 1}
```

```{r grade_conditions-hint-1}
# hint text
"Function should take a parameter and add 1 to it"
```

```{r grade_conditions-check}
gradethis::grade_conditions(
  gradethis::pass_if(~ .result(3) == 4),
  gradethis::pass_if(~ .result(-1) == 0),
  gradethis::fail_if(~ .result(10) == 11)
)
```
``
```

# Chunk names matter

```
```{r chunkLabel, exercise=TRUE}
log(____(____))
```

```{r chunkLabel-hint-1}
# hint text
"Exponentiate with the `exp` function."
```

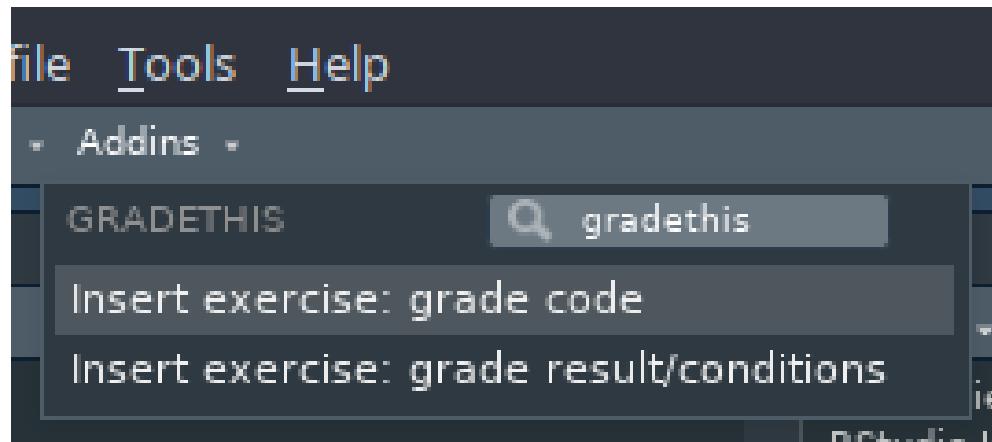
```{r chunkLabel-solution}
# solution code
log(exp(3))
```

```{r chunkLabel-check}
# check code
gradethis::grade_code()
````
```

- The hints/solution/check blocks use the same base name for association
- `knitr` chunks need to be unique

# gradethis chunk addins

- Creates random chunk names



```
```{r qhdcovmzxhuycxw, exercise = TRUE}
# student code
```
```
```

```{r qhdcovmzxhuycxw-hint-1}
# hint text
```
```
```

```{r qhdcovmzxhuycxw-solution}
# solution code
```
```
```

```{r qhdcovmzxhuycxw-check}
# check code
gradethis::grade_code()
````
```

# Be careful with ==

- <https://daniel.rbind.io/2019/08/06/inconsistencies-with-in-r/>
  - "inconsistency", "improper documentation", "bug"

```
# only difference is the last parameter
u <- quote(f(x12345678901234567890123456789012345678901234567890, 1))

s <- quote(f(x12345678901234567890123456789012345678901234567890, 2))
```

```
u == s
```

```
## [1] TRUE
```

```
identical(u, s)
```

```
## [1] FALSE
```

- Take away: use `identical` or `all.equal` instead of `==` when doing (non-vectorized) comparisons.
  - Vignette in PR: <https://github.com/rstudio-education/gradethis/pull/102>

# Dev guide: `learnr` vs `gradethis`

- `learnr` is responsible for:
  - Expression capturing
  - Data reporting
  - How many questions did my student get "correct"?
  - How much time did my students take for each question or for the entire document?
- `gradethis` is responsible for:
  - Checking the student solution
  - Correct/Incorrect message + praise/encouragement

# Dev guide: `learnr` checker

- `learnr` is a complex piece of software that needs a deep understanding of `knitr`, `rmarkdown`, and `shiny`.
  - Nischal Shrestha (2020 Intern): <https://education.rstudio.com/blog/2020/07/learning-learnr/>
- All of the exercise components in the `learnr` document are passed into the `exercise.checker`.
  - This is an entry point if you want to write your own `learnr` checker
- To enable exercise checking in your `learnr` tutorial:

```
```{r setup}
library(gradethis)
tutorial_options(exercise.checker = gradethis::grade_learner)
```
```

- The `grade_learner` function:
  - The function `learnr` passes all the chunk code into
  - Passes into the corresponding `gradethis::grade_` function

# Dev guide: Pipes

```
df %>% dplyr::select(col1)
```

unpipes into

```
dplyr::select(df, col1)
```

# Standalone code grader: result

```
library(gradethis)

student_solution <- 5

graded_result <- gradethis::grade_result(
  pass_if(5, "You also got a value of 5!"),
  learnr_args = list(last_value = student_solution)
)

## Loading required namespace: testthat

graded_result

## $message
## Swell job! You also got a value of 5!
##
## $correct
## [1] TRUE
##
## attr(",")
## [1] "grader_graded"
```

# Standalone code grader: code

```
student_code <- quote(sqrt(exp(3)))
solution_code <- quote(sqrt(log(2)))
```

```
graded_code <- gradethis::grade_code(grader_args = list(
  user_quo = student_code,
  solution_quo = solution_code
))
graded_code
```

```
## $message
## I expected a call to log(); what you wrote was interpreted as exp(3). Try it again. Perseverence is the key to
## 
## $correct
## [1] FALSE
## 
## attr(,"class")
## [1] "grader_graded"
```

# Internship: eval\_tidy example

```
form <- ~ x ^ 2 # capture an expression + environment
str(form)

## Class 'formula'  language ~x^2
##   ..- attr(*, ".Environment")=<environment: R_GlobalEnv>

form[[2]] # just get the expression

## x^2

rlang::eval_tidy(
  expr = form[[2]], # evaluate the expression with data + environment
  data = list(x = 15),
  env = .GlobalEnv
)

## [1] 225
```

# Internship: eval\_tidy in practice

[https://github.com/rstudio-education/gradethis/blob/master/R/evaluate\\_condition.R](https://github.com/rstudio-education/gradethis/blob/master/R/evaluate_condition.R)

During `gradethis::grade_result`:

```
switch(condition$type,
  "formula" = evaluate_condi_formula(condition$x, learnr_args$last_value, learnr_args$envir_prep),
  "function" = evaluate_condi_function(condition$x, learnr_args$last_value),
  "value" = evaluate_condi_value(condition$x, learnr_args$last_value)
)

evaluate_condi_formula <- function(formula, user_answer, env) {
  form_result <- rlang::eval_tidy(
    formula[[2]],
    data = list(.result = user_answer, . = user_answer),
    env = env
  )
  return(form_result)
}
```

# Python!

- pygradethis by Nischal Shrestha (Current RStudio intern)
  - <https://github.com/nischalshrestha/pygradethis>
- Can use Python + `learnr` to check results, but checking the result is more involved.
  - Python does not have a `quote` function
  - Working with code expressions involve quoting things as strings and using `ast.parse()`

**Still in (very active) development**

```
grade_code("2 + sqrt(log(2))", "2 + sqrt(log(1))")
```

I expected 1, but you wrote 2 in `log(2)` at line 1.

# Thanks!



@chendaniely

slides: [https://github.com/chendaniely/rstatsnyc\\_2020-learnr\\_gradethis](https://github.com/chendaniely/rstatsnyc_2020-learnr_gradethis)

pdf: [https://speakerdeck.com/chendaniely/rstatsnyc\\_2020-learnr\\_gradethis](https://speakerdeck.com/chendaniely/rstatsnyc_2020-learnr_gradethis)