Grading Code with gradethis

satRday 2020

Daniel Chen @chendaniely

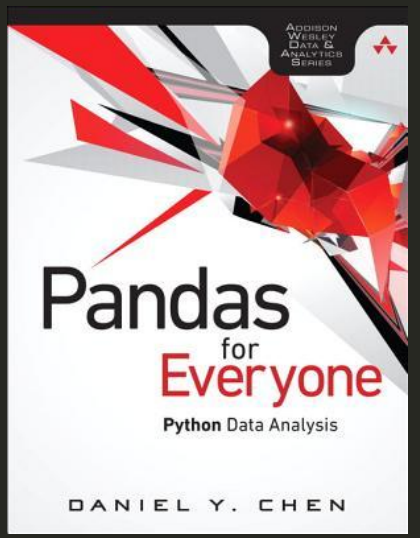
Virginia Tech

2020-03-28

hi!

I'm Daniel

- PhD Student: Virginia Tech
  - Data Science education & pedagogy
  - Medical, Biomedical, Health Sciences
  - Advisor: Anne Brown, PhD
    - https://www.databridge.dev/
    - https://bevanbrownlab.com/
- Inten at RStudio
  - `gradethis`
  - Code grader for `learnr` documents
- Author:



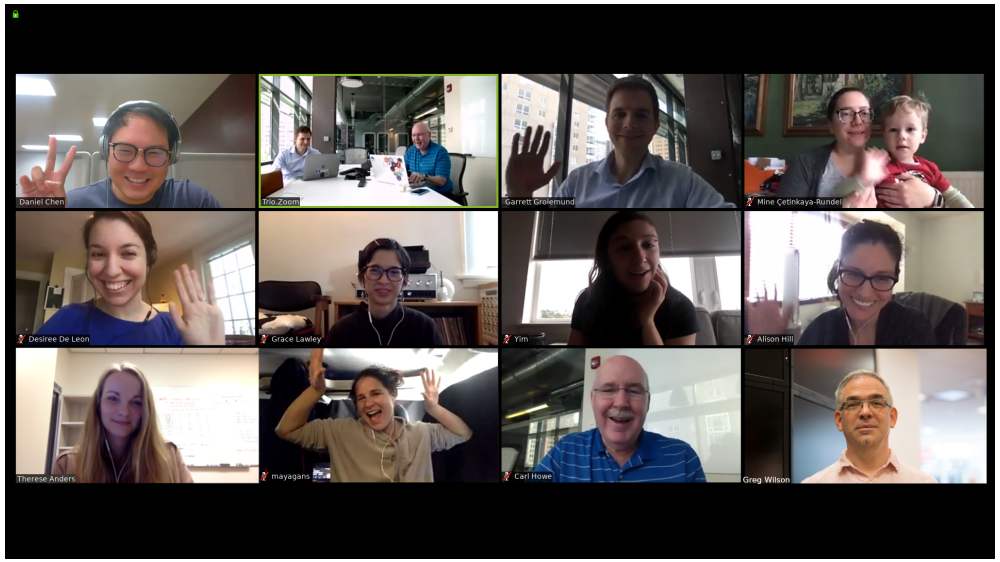
RStudio Internship

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

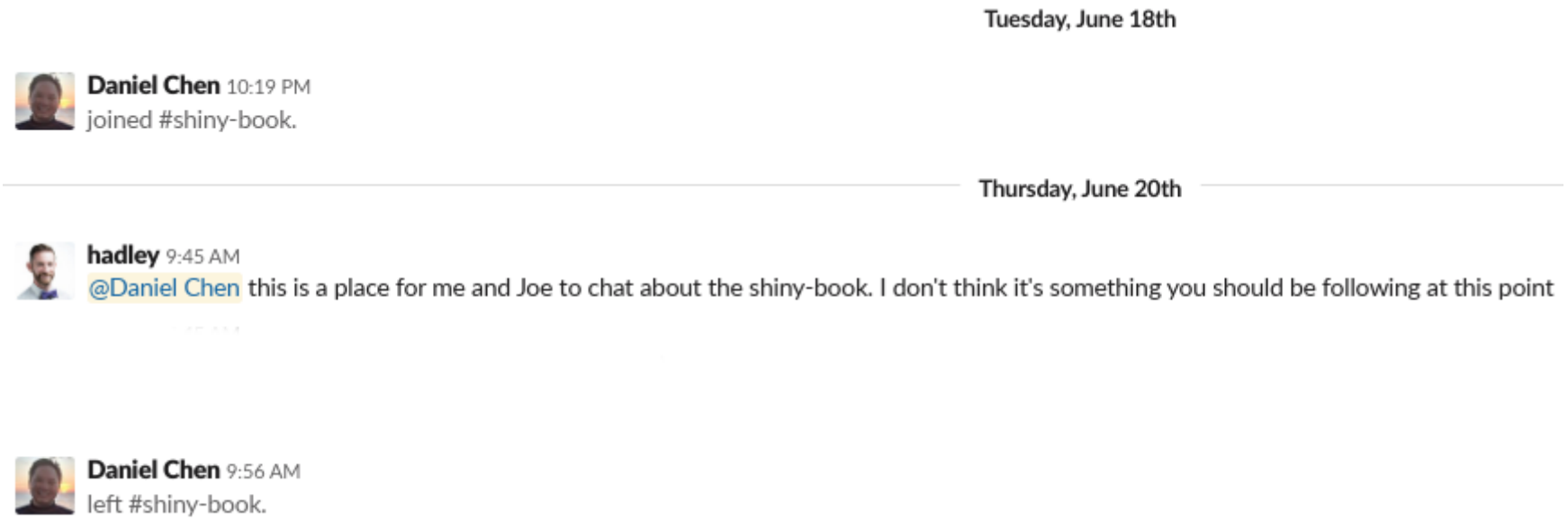
Education Team



Shiny Team

Join all the slack channels!

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



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

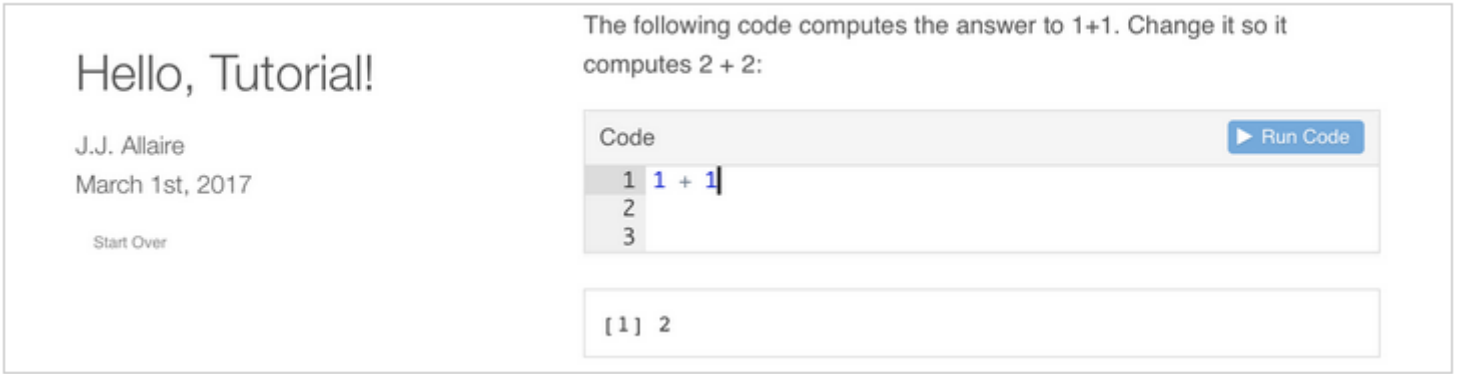
````
---
title: "Hello, Tutorial!"
output: learnr::tutorial
runtime: shiny_prerendered
---

```{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
```
````

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



Sortable

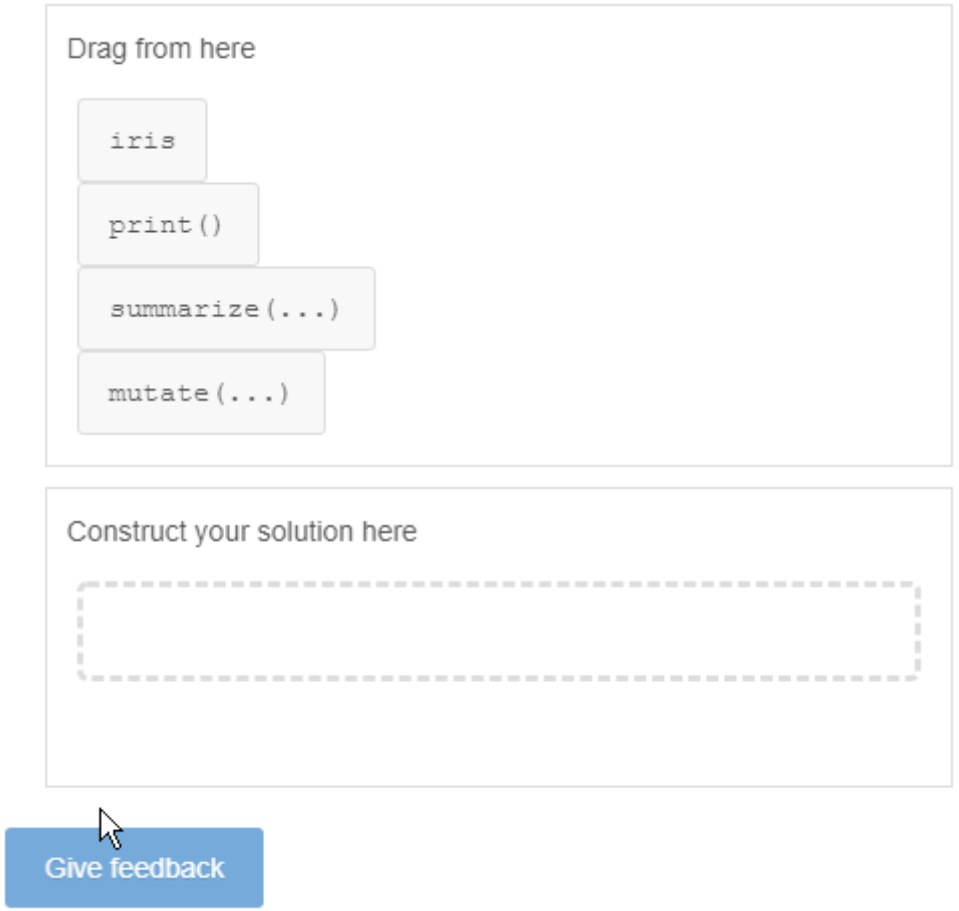
- More `learnr` examples: https://rstudio.github.io/learnr/examples.html
  - `sortable`: https://andrie-de-vries.shinyapps.io/sortable_tutorial_question_rank/

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))
)
```

```
## ▆─log
## └─▆─exp
##   └─3
```

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

```
## ▆─log
## └─▆─exp
##   └─2
```

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

```
## ▆─log
## └─▆─log
##   ├─2
##   └─base = 10
```

Gradethis

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

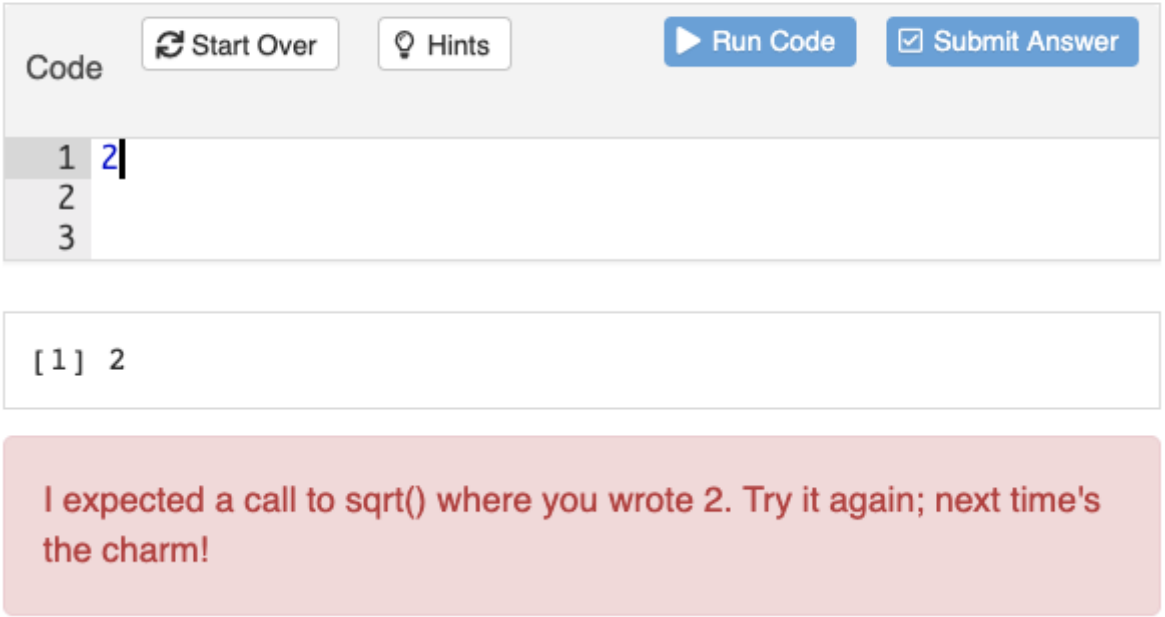
- Auto grading system that can be used to grade `learnr` exercises.
  - Check the answer
  - Check the code (using the AST)
- In `learnr`, each exercise is completely independent from the others
- What makes it unique is its ability to give formative feedback to the student.
- `leanr` but why not `grader`?
  - 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.



`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.
````markdown
```{r addition, exercise=TRUE}
log(_____(_____))
```
````

And want the student to enter

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

learnr + gradethis::grade_result()

````
```{r addition, exercise=TRUE}
log(_____(_____))
```

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

```{r addition-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 addition, exercise=TRUE}
log(_____(_____))
```

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

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

```{r addition-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 addition, exercise=TRUE}
log(_____(_____))
```

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

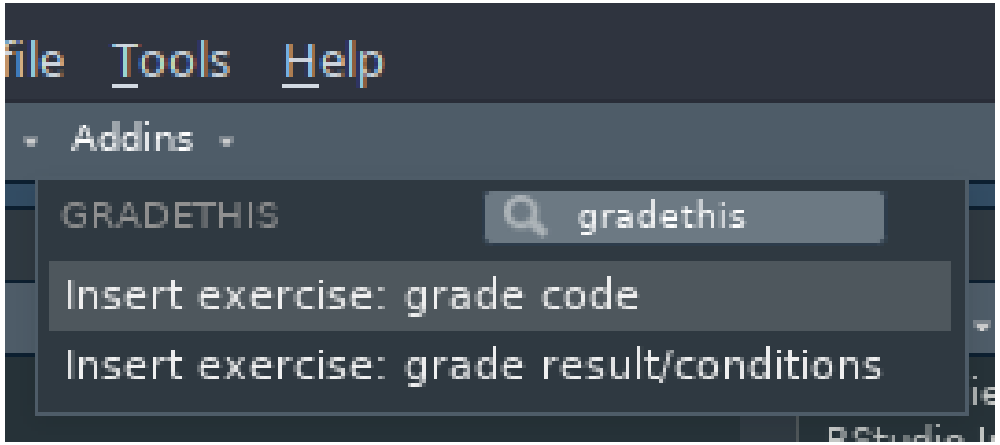
```{r addition-solution}
# solution code
log(exp(3))
```

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

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

`gradethis` chunk addins

- Creates random chunk names




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

____
```

```{r qhdcovmzxfhuycxw-hint-1}
# hint text
""
```

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

```

```{r qhdcovmzxfhuycxw-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 `idential` 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`.
- 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_learnr)
```
````

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

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
## Bravo! You also got a value of 5!
##
## $correct
## [1] TRUE
##
## attr(,"class")
## [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() where you wrote exp(3).  Let's try it again.
##
## $correct
## [1] FALSE
##
## attr(,"class")
## [1] "grader_graded"
```

User -> Dev

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

During `gradethis::grade_result`:

```
switch(condition$type,
           "formula" = evaluate_condi_formula(condition$x, learnr_args$last_value, learnr_args$
           "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)
}
```

Internship: `lapply` good

```
vec <- 1:10
```

```
lapply(vec, log, base = 10)
```

```
## [[1]]
## [1] 0
##
## [[2]]
## [1] 0.30103
##
## [[3]]
## [1] 0.4771213
##
## [[4]]
## [1] 0.60206
##
## [[5]]
## [1] 0.69897
##
## [[6]]
## [1] 0.7781513
##
## [[7]]
```

Internship: `sapply` bad

```
# simplify this value into a vector, if you can't just give me a list
sapply(vec, log, base = 10)
```

```
##  [1] 0.0000000 0.3010300 0.4771213 0.6020600 0.6989700 0.7781513 0.8450980
##  [8] 0.9030900 0.9542425 1.0000000
```

Internship: `vapply` better

```
# I want a numerical vector back where each element only contains 1 element
vapply(vec, FUN = log, FUN.VALUE = numeric(1), base = 10)
```

```
##  [1] 0.0000000 0.3010300 0.4771213 0.6020600 0.6989700 0.7781513 0.8450980
##  [8] 0.9030900 0.9542425 1.0000000
```

Thanks!

@chendaniely

slides: https://github.com/chendaniely/2020-03-28-satRday-gradethis

pdf: https://speakerdeck.com/chendaniely/satrday-grading-code-with-gradethis