From: **lizbot** deid@appacademy.io
Subject: Evening Announcements for 16 Week - W1D1
Date: August 31, 2020 at 9:09 PM
To: 2020-08-31-ny@appacademy.io
Cc: instructors-ny@appacademy.io

Hello folks! Congratulations on completing your first day!

This will be the first in the series of daily emails to summarize the major takeaways of the day's projects and general announcements. Use this as a (non-comprehensive) guide for studying and reference as the course goes on. Today's has a lot of info, as you all have been exposed to quite a number of topics!

Key Points

Ruby Style

- Ruby has implicit returns, meaning the last expression that is evaluated in the method will be the return value! (You can still return early using the `return` keyword.)
- When a method takes no arguments, it is preferred to invoke (or run) the method *without* parenthesis, i.e. (preferred) `print_banana` vs. `print_banana()`
- You can write `if` statements on one line if there is no related `elsif` or `else`
- There are many helpful methods that make Ruby easier to read, such as: `even?`, `odd?`, `zero?`, `first`, `last`
- Enumerables are methods that make performing repetitive operations (i.e. iterating over an array) clean and straightforward. Some include: `all?`, `any?`, `none?` `one?`, `count`, `sum`, `max`, `min`; and we prefer to use these over loops when we can.

Methods

- By using *default arguments*, we can make arguments to methods optional. Given: `def my_method(vegetable, fruit="banana")`, `my_method` must be invoked with at least one argument (`vegetable`), and given a 2nd optional argument (`fruit`). If we don't pass in the optional argument, the argument will be implicitly passed in with whatever value is set as the default (in the example, `fruit` will be set to the string `"banana"`).
- Ruby methods can be defined to take in *options hashes*. They should be the *last* parameter in the method definition. When calling a method that takes in an options hash as an argument, you can omit the curly braces , `{}`, for the hash.
- The splat operator (`*`) allows for flexibility with the number of arguments a method can accept. By adding the splat operator to the last parameter of a method, we can access any additional arguments given to a method as an array!
    - Additionally, the splat operator can be used to decompose (unpack) elements of an array; think of it like it's removing the `[]` of the array to give us a comma separated list of items.
- In Ruby, methods have their own *local scopes*. This means the code in the method has access to only the variables defined in this method. Global variables have global scope, meaning they can be accessed anywhere, but you can run into naming collisions and confusing code if you rely on these! Note that blocks do not have their own scope (they are part of the scope of the method they are in), so we can reference variables defined outside of the block.

Inject

- `inject` is a powerful array method that allows us to get a computed value(referred to as an *accumulator*) that changes as we run some operation on each element of the array. The block to inject takes in an accumulator (the value we want) and an element. The block will be run for each element; note that *the accumulator will be set to the last expression evaluated in the block* for each iteration! This allows us to *accumulate* a value over time as we look at each element.

Variables and References

- Be careful with variables and references in Ruby! Remembers, variables point to an address in memory. If you ever need to check if two variables reference the same object (same address in memory) or different objects, you can use the `object_id` method.
- A common issue to run into is when we have a 2D array (matrix). If we try to make an 3x2 matrix with `Array.new(3, Array.new(2))`, (size, default value) we have said "Let's make an array of size 3; and each element's default value will be equal to this array of size 2". This has made each element reference the *same* array of size 2! What we really want to do is `Array.new(3){Array.new(2)}`, which says "Let's make an array of size 3, and each element's default value will be set to running the code `Array.new(2)`". In this case, `Array.new(2)` is run for each element's default value, meaning each element is a *different* array of size 2.

Raising Errors and Error Handling

- Sometimes code you write doesn't support a certain behavior; like `10 / 0` in Ruby. In cases like these, an *error* will be *raised*. If unhandled, these will stop the execution of your program before it's finished! You can handle these by having a `begin...rescue`. `begin` should contain code you'd like to run that may raise an error. If any code in the being raises an error, `rescue` will "rescue", or prevent, the program from stopping.

- There might be certain scenarios where *you* want to raise an error! You can do this by using `raise "Your error message"`

Nightly Workflow

- **IMPORTANT: Submit your daily report on Progress Tracker before 9am the next day.** Make sure to do this every night!
- Do the **Homework** for the next day (i.e. you will be doing the Homework for W1D2 tonight!)
- Watch the solutions (note, only do this if you finished the projects before 6pm, or after 6pm; don't look at these during the project for help!)
    - Note: You can speed up the video by clicking on the *Gear Icon,* and clicking on *Speed*
- If you have time after this, *then* you can work on finishing the day's projects

Do everything under the "Homework" section before the start of that day!

Evening Announcements

- **CHECK-IN PROCEDURE**: A TA will clear the check marks in the participants window, that is accessible through the toolbar along the bottom of your Zoom screen. In order to be checked-in, you **must** have your **camera on and throw a green check mark.**
    - Check-in times are **9:00am, 1:30pm, and 4:00pm.**

- With pairing, make sure you understand all of the code you and your pair write. If there's ever anything you're not comfortable with, ask your pair to stop and explain. It's very important to make sure you understand all the material. Equally important is that you and your pair get to practice the very important skill of explaining your ideas.
- Reminder on Pair Programming
    - **Switch roles every 15 minutes**! Set a timer and abide by it. Don't "wait til we finish this part, method, etc."
    - Both pairs should be on the same page in terms of the plan so switching should be seamless.
    - If you aren't on the same page with the general idea of how to solve, **pause the timer** and come up with a plan together (work out any gaps in understanding as needed)
    - **Never open the solutions** while you are pairing! If you get stuck, do some testing together, google your issue, and if you are still stuck, ask your awesome instructors for help by clicking the "Ask A Question" button on Progress Tracker!

Pairing Roles (excerpt from this link):

Driver

- Write the code according to the navigator's specification
- Listen intently to the navigators instructions
- Ask questions wherever there is a lack of clarity
- Offer alternative solutions if you disagree with the navigator
- Where there is disagreement, defer to the navigator. If their idea fails, get to failure quickly and move on
- Make sure code is clean
- Own the computer / keyboard
- Ignore larger issues and focus on the task at hand
- Trust the navigator - ultimately the navigator has the final say in what is written
- You are writing the code

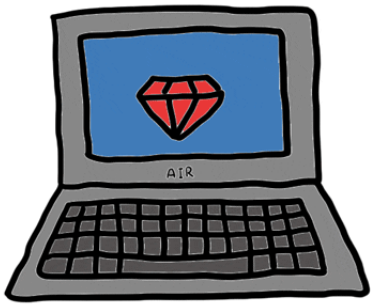
Navigator

- Dictates the code that is to be written - the 'what'
- Clearly communicates what code to write
- Explains 'why' they have chosen the particular solution to this problem
- Check for syntax / type errors as the Driver drives
- Be the safety net for the driver
- Make sure that the driver sticks to the small task at hand
- Outline and note down high level tasks/ issues
- Ongoing code review
- Pay attention
- Wait until the task is complete to bring up design / refactoring issues

Both

- Actively take part in programming
- Aim for optimal flow - avoid trying to be 'right'
- Embrace your role
- Intervene if your pair is quiet
- Communicate, communicate, COMMUNICATE!
- Sync up frequently to make sure you are on the same page
- Don't hog the keyboard

- High-five every time a test passes



Keep up the great work everybody! You can do this!

- Lizbot