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. 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 the following:
* def fav\_fruit\_and\_veg(fruit, veg = "broccoli")
* sent = "My favorite fruit is #{fruit} and my favorite vegetable is #{veg}"
* return sent

end

* + The method fav\_fruit\_and\_veg must be invoked with at least one argument (*fruit*) and can be given a 2nd optional argument (*veg*).
  + If we don't pass in an optional second argument, the parameter *veg* will be implicitly set equal to (in this case) the string "broccoli"
* 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 in the array. NOTE: 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! Remember, 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)) 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! The daily reports become available at 6pm PDT.
* Do the Homework for the next day (i.e. you will be doing the Homework for W1D2 tonight!)
* Watch the solutions and video walkthroughs (only do this 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

**Behavioral Health Survey**

* Please take a few moments to complete the student survey linked here: [App Academy Behavioral Health Survey](https://docs.google.com/forms/d/19JtVAr0QT-n-tftfb087v9jPzsZjj0izBkk_IJce_Ug/edit)
* If you have other questions, reference [this FAQ](https://docs.google.com/document/d/1vf3CWzGWs1R3IC0qfGg0gA3C92I-LOPKuAff1qM0Bqk/edit) or feel free to reach out to anyone on the Instruction Team.

**Evening Announcements**

* Check-in times are **9:00am**, **1:30pm**, and **4:00pm**. Make sure your camera is on and are ready for roll call at these check in times on Zoom. Please make sure to set up your alarms 5 minutes beforehand in order to ensure you are not late. If you will be late for whatever reason, you MUST slack an instructor BEFORE check-in.
* Make sure you understand all of the code you write. If there's ever anything you're not comfortable with, ask an instructor! It's very important to make sure you understand all of the material.
* Debugging is just as important as coding. Make sure you read your error messages, google them, and spend time breaking your code so you know how you would handle a problem on an assessment or in the real world.
* Make sure to not check solutions until after the day is over! If you get stuck on a problem for longer than 20 minutes, please ask a question on Progress Tracker.
* If you haven't recently updated Zoom, please follow the instructions [here](https://support.zoom.us/hc/en-us/articles/201362233-Upgrade-update-to-the-latest-version) in order to do so.
* Here are [Cohort Resources (General)](https://github.com/appacademy/cohort-resources/tree/master), please bookmark this page because it has a lot of valuable information and resources!