- Memory Debugging Tips 'n' Tricks
  - Problems
    - Retain Cycles
      - closures
      - delegates
    - Abandoned Memory
      - global memory keeping a strong reference to something it shouldn't
  - Tools
    - Visual Memory Debugger
      - Left pane shows all objects in memory and how many instances are currently alive.
        - You can filter it down to just the objects from your project with the icon in the bottom right of the pane that looks like a "new document" icon. That is also a text box down there in which you can just filter the list down by name. There's another button in there with an ! on it, that filters the list to just objects that are leaked.
        - Selecting an instance shows it in the graph in the center.
      - Center pane shows the memory graph of the selected object, allowing you to see all of the objects that currently have strong references to the object.
      - Inspector on the right shows back trace for selected object, which can be clicked on to go to that line of code in your project!
    - Memory Usage Graph in Debug Session Pane
      - Keep an eye on this while debugging in general. If you know your memory footprint's normal size, then when it's larger than normal, you'll know you have a problem to investigate.
  - Solutions
    - Retain Cycles
      - Capture Groups
        - Can be applied to any object being captured by a closure (but usually it's just self)
        - [weak self] - for when the closure might exist and run after self has been deallocated
          - basically makes self an optional in the closure
        - [unowned self] - for when the closure can't be run unless self still exists
          - basically makes self an implicitly unwrapped optional in the closure
      - Declare delegate properties as weak vars. This requires the protocol to have the `class` limiter, so that only reference types can implement it.
    - Abandoned Memory
      - Use the tools to find where the errant strong references are coming from and make them weak references.
      - If you're not finding the issue with the above tools, some strategically placed print statements in `deinit`s might help.