1. What have you learned recently about iOS development? How did you learn it? Has it changed your approach to building apps?

I came to iOS development because I had an idea for a very specific app in 2012, before Swift existed. Objective-C looked too hard for a beginner, so I thought I would learn to make my app as a webpage. By the time Swift was introduced, I was very familiar with Python, which has a lot in common with Swift, so I dived into iOS development and fell in love. And just like really being in love, with every new thing I learned, I thought, "Oh great! Let's find out more about that! Wait -- what's this? Let me look into this cool new thing!" so after a year, I had learned a lot of Swift, but I hadn't finished or polished any projects, and I didn't have much to show for it.

I decided to sign up for a Udacity iOS Developer Nanodegree in February, and in the last three months, I feel like I have leveled up from being a "Swift hobbyist" to a real iOS developer. I had all the basic building blocks of iOS development in my notes and scattered projects, but in constructing the six apps required for the Nanodegree, I was able to polish my knowledge and put all of the many concepts I have learned together. I'd used frameworks like UICollectionView, MapKit, NSURL and Core Data before but now I know how to combine them, delve deep into what they can do, and ensure that my code is clean, reusable, and safe. Now I know how to structure an app properly with an MVC or an MVVM pattern, I know how to use singleton patterns to pass information around my app, and I know how to build an app from the ground up using Test Driven Development.

Now that I’ve completed my Udacity iOS Nanodegree, I'm not just learning to be an iOS developer anymore -- I am one!

2. Can you talk about a framework that you've used recently (Apple or third-party)? What did you like/dislike about the framework?

Is there any framework more important and more fun than UIKit? Before coming to Swift and iOS Development, I did some web design and a tiny bit of software design with Python, and I can honestly say that UIKit is the framework that made me fall in love with the iOS SDK and want to be an iOS Developer.

The really special thing that Apple has done with UIKit is take all the many, many things you can do to make your app look great, and give them to you without any of the hassle you get in web and software development about the "nitty-gritty" -- the problems of making sure everything is exactly sized right and looks the same on every single version of every single web browser, the challenges of changing photo resolution sizes for every single type of screen, the headaches of remembering what CSS is supported 100% by what versions of what browser -- those things are gone, and all that remains is a realm of possibility where you can use the strengths of UIKit to make all your design dreams come true.

AutoLayout! What a magical invention! You can clearly see your design and add constraints that will make it look great on all Apple's devices with a few mouse clicks! Stack Views! Vary for traits -- your design can even make major shifts for device sizes! Give Apple three sizes of images and your images will look fantastic on every device! And if Main.Storyboard is still too much hassle for you? UIKit gives you simple and easy ways to do exactly the same AutoLayout in code! Add in the incredibly simple to use Animation framework and every element in your view can move, spin, drop and provide the user with as much joy and creativity as you want!

When I discovered UIKit, it felt exactly the same as opening a box of Legos as a child -- Apple has given developers such an easy toolbox to play with where the only limits are their own imaginations. When you're freed from the hassle of the smaller things, you can achieve the bigger ideas you have in mind and make more complicated apps that do much more impressive tasks and reach a wider audience.

3. Describe how you would construct a Twitter feed application (here is an example of Udacity's Twitter feed) that at minimum can display a company's Twitter page. Please include information about any classes/structs that you would use in the app. Which classes/structs would be the model(s), the controller(s), and the view(s)?

Twitter provides a TwitterKit for iOS that can make constructing a Twitter feed application incredibly easy. I would use Cocoa Pods to install the TwitterKit framework in my app and then structure my app like this:

MODEL: TwitterRequestModel (for the TwitterAPI call)
TweetModel (for the Twitter Stream and Tweets received)
DataManager (To format and control the data received from the API).

To display a company's Twitter page, I'm going to need at least two models: a tweet model that outlines the structure of a group of all the company's tweets as well as what makes up a single tweet, and a client model that contains all the information I need to use Twitter's REST API service to make a network call to the Twitter API and load a set of all the company's tweets in a JSON feed. The Client Request model can be a struct, since it will hold simple values and methods. A struct would probably do for the Tweet model given TwitterKit's simplicity and the idea that the initial functionality for this app would be just to display this company's tweets. The TwitterClient model would also include a class with methods used to control the basic URLSession, fetch requests, and JSON Serialization. I would also probably opt to include a DataManager class to handle the Twitter data I receive from the API, to process it so it's ready for the View Controller, and also to implement the ability to have an infinitely scrolling UITableView of tweets on the main display page.

CONTROLLERS: UITableViewController (to display all the tweets)
CloseUpViewController to focus on a single selected tweet.

I would need two basic controllers for this app: a UITableViewController to display the full list of the company's tweets, and another UIViewController to display a detail view of a selected tweet. TwitterKit comes with functionality that makes this relatively easy: TWTRTweetViewStyleCompact is designed to go inside a UITableViewController, and TWTRTweetViewStyleRegular is designed for a detail page.

VIEW: TWTRTViewStyleCompact custom UITableViewCell Class
TWTRTViewStyleRegular custom UITableViewCell Class
Reachability
AlertManager

The view portion of my app would need several classes. I would need to integrate TwitterKit's TWTRTweetTableViewCell in a custom UITableViewCell class to display the TWTRTViewStyleCompact and TWTRTViewStyleRegular tweets. Since I'm making an API call, I would need an AlertManager class to handle UIAlertControllers to display information to the user about any network issues or any problems displaying the company tweets, and a Reachability class to check any connectivity issues the user might be having.

4. Describe some techniques that can be used to ensure that a UITableView containing many UITableViewCell is displayed at 60 frames per second.

The great news about attacking the problem of making sure a UITableView displays at 60 frames per second is that Apple has already given developers a huge advantage -- if you smartly and correctly follow the Apple documentation guidelines for UITableView, UITableViewDelegate, and UITableViewDataSource, you should easily achieve the 60 frames per second display rate.

The most important element in keeping a sleek, fast table view design is using dequeueReusableCellWithIdentifier. This ensures that only the table view cells that are currently visible exist in memory, and it allows cells not being used to be formatted for use while still off-screen, and then disposed of in memory when they go off-screen again. This way the cells are recycled on the fly instead of created all at once, and it keeps the table view from being bogged down with excessive loading.

Another great way to keep the frame rate high on a UITableView is to implement the logic you need to display the cells in the tableView(willDisplay) cell method, instead of the tableView(cellForRowAt) method. This keeps any of the heavy loading off screen, giving it plenty of time to be calculated before the cell comes into view. Putting too much logic inside tableView(cellForRowAt) means that all of the processing will to happen at view time, which will slow down the frame rate.

It's also a good idea to keep any costly memory operations outside of a UITableView and UITableViewCell. Any heavy data calculations should be done well in advance and only the results should ever be visible to the table view during cell creation. Table cell dimensions should be kept as simple as possible, since running the heightForRowAtIndexPath method can be costly. Also, it's best to keep any animations out of the UITableView, as animation can greatly slow down the table view frame rate.

5. Imagine that you have been given a project that has this ActorViewController. The ActorViewController should be used to display information about an actor. However, to send information to other ViewControllers, it uses NSUserDefaults. Does this make sense to you? How would you send information from one ViewController to another one?

NSUserDefaults should never be used to store information and pass it from one portion of an app to another. NSUserDefaults is best used for storing things like user preferences -- would the user like the sound on or off, a white or a black background -- and not data that's needed for the main functionality of the app. A simple struct or class can be created that can retain and pass data to other parts of the app, or in an instance where data might fetched and re-fetched from an API, Core Data would be a much better choice to persist data.

There are two simple ways to pass actor data to this view controller without using a persistent storage method that might be useful: create a single instance of actor and refer to it throughout the app, or initialize an actor instance from a class or a struct, and pass that through the app inside the prepare(forSegue) method.

```
struct Actor {
   var actorName: String
   var actorImage: UIImage?
   var actorBio: String?

   init(name: String, image: UIImage?, bio: String?) {
      self.actorName = name
      if let image = image {
         self.actorImage = image
      }
      if let bio = bio {
         self.actorBio = bio
      }
   }
}
```

In a simple app, where you're going to need this actor's information over a series of views, you could create a singleton instance of an actor inside AppDelegate:

```
let actor = Actor(name: "Harrison Ford", image: UIImage(named: "Han Solo"), bio: "Harrison was a carpenter before he became an actor.")
```

And then reference this instance inside the ActorViewController like this:

```
let appDelegate = (UIApplication.shared.delegate) as! AppDelegate
let actor = appDelegate.actor
```

Another way to do this in a more complicated app, where you may be retrieving many actor's information from an API and going back and forth from a larger list to this detail view of an actor, would be to initialize the actor as an instance of the struct inside the previous view controller, as above, and then pass it to the ActorViewController inside prepare(forSegue):

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // send selectedActor information to ActorViewController detail display
    if segue.identifier == "showActorDetails" {
      let actorViewController = segue.destination as! ActorViewController
      actorViewController.actor = actor
    }
}
```

If there is a good reason to persist the data in the app, an Actor Entity inside of a Core Data Model would be a much better choice than NSUserDefaults.

6. Imagine that you have been given a project that has this Github ProjectViewController. The GithubProjectViewController should be used to display high-level information about a GitHub project. However, it's also responsible for finding out if there's network connectivity, connecting to GitHub, parsing the responses and persisting information to disk. It is also one of the biggest classes in the project. Follow-up question:: How might you improve the design of this view controller?

Let's take things that don't have to do with presenting the current view outside of our massive GitHubProjectViewController. Code related to finding out if there's network connectivity, connecting to GitHub, parsing the responses and persisting information to disk doesn't belong inside the view controller itself.

Create a new CLIENT group:

First of all, let's take all the code related to making network queries and move it to a class that deals specifically with the API call in a Client group.

```
class GitHubClient {

  func fetchNameAndCompletionPercentage() {
    // code currently inside the viewDidLoad
  }

  func loadAllIssues() {
    // code currently inside the viewDidLoad and postNewIssueButtonPressed
  }

  func postNewIssue() {
    // code currently inside the postNewIssueButtonPressed method
  }

  func loadMostRecentComments() {
    // code currently inside the tableView(didSelectRowAtIndexPath)
  }

  func loadContributors() {
    // code currently inside the collectionView(didSelectItemAtIndexPath)
  }

}
```

We may want to make a separate DataManager class to format the data received before it's passed back to this GitHubProjectViewController. It's unclear here if there is logic being performed on the data inside tableView (cellForRowAtIndexPath) or collectionView(cellForRowAtIndexPath), but if there is, it should be extracted and the code inside this method should simply fill the cell with information, not perform any logic on that information.

Create a Model Class or group:

Any code related to persisting information should go inside a model class. The DataManager class can interact with this model and format the information before passing it back to the GitHubViewController, or, if Core Data or another database is being used to persist information, there should be an entirely separate class of files in our Model group to deal with the persistence code.

Create a View group:

If there is a significant amount of styling information being performed on the cell inside tableView(cellForRowAtIndexPath) or collectionView (cellForRowAtIndexPath), that can be removed and placed inside a class in the View group that deals specifically with a custom UITableViewCell or a custom UICollectionViewCell.

Let's take the alert views and also export them to their own AlertManager class inside our View group.

```
class Alerts {

  func displaySuccessOrFailure() {
    // code currently inside the postNewIssueButtonPressed method
  }
}
```

Create a SupportingFiles group:

Any code related to finding out if there's network connectivity should be placed in a new InternetConnectivityManager inside a SupportingFiles group.

7. If you were to start your iOS developer position today, what would be your goals a year from now?

Even though I have an iOS Development Nanodegree and an app in the App Store, I'm currently still very much a student of Swift and iOS development. I have much more to learn about the practical details of building and maintaining a large-scale app, and I am really looking forward to being part of a great development team so that I can learn from more experienced iOS developers around me. But in one year, I see myself as having made the leap from student to teacher, and having moved to a role where I can also contribute new tips and techniques to a team environment and coach more junior iOS developers.

Swift has a really dedicated community of coders who love to share information. I use the community mostly as a resource now, but it's time for me to also start contributing to it. I want to be able to pass on all the cool things I know to people who are just learning. I want to take the new framework I just heard about on a podcast and implement it in my own app, and then take it to my LearnSwift group and introduce it to new people. I want to be the person newbies come to when they need to iron out the kinks in their customized UICollectionViewCells. I want to not just read answers on Twitter and StackOverflow, but also provide them.

Career-wise, in a year, I'd like to be in the middle of a very exciting development team environment. I still want to learn as much as I can from those with more experience, but I also want to be the person who introduces my team to a new time-saving trick in AutoLayout and helps the junior developers learn Test Driven Development. One of the most fun things about working in iOS development is the fast-paced, ever-changing team dynamic, and I can't wait to be a part of it.