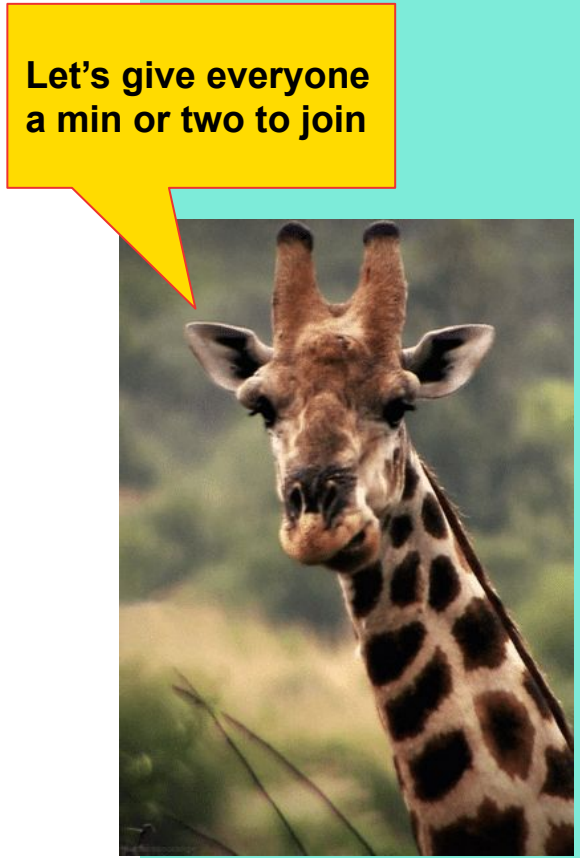
Whilst we wait....

In chat write:

Do you have a pet? What kind of pet? Name? If you do not have a pet, what's your favourite animal?



 © 2021 General Assembly

Welcome to Modern Engineering!

Unit	Summary
Lesson 1 **Unit 1: Command Line Interface with Linux & Bash** **Distributed Version Control with Git & GitHub**	Practice working with command line and reinforce Unix Shell fundamentals Work with both local and remote Git repositories and with features that enable team collaboration
Lesson 2 **Cloud Providers (3 hours)**	Deep dive into moving to cloud.
Lesson 3 **Cloud Infrastructure (3 hours)**	Understand the benefits of adopting a cloud-based infrastructure and the different models of delivering cloud services.
Lesson 4 - 6 **Containers with Docker and Kubernetes (9 hours)**	Use containers and understand how they fit into modern development workflows.
Lesson 7 **Microservices Architecture (3 hours)**	Explain the benefits of microservice architecture and build scalable microservice applications.
Lesson 8 & 9 **Continuous Integration with Jenkins (6 hours)**	Automate your testing and deployments by using continuous integration pipelines in Jenkins.
Lesson 10 **Extreme Programming (3 hours)**	Use XP workflows to manage development workflows more efficiently.
Lesson 11 **Test Driven Development (3 hours)**	Increase test coverage for your code to build and maintain confidence with unit and integration tests.
Lesson 12 **Capstone Projects (3 hours)**	Completed throughout the delivery with a presentation on the last session.

Remember Your To Do List

- Be engaged and ask questions
  Start every session with your microphone **muted** and **camera on**
- Communicate with us
- Don’t be afraid of breaking things
- Don’t worry about writing things down
- Try not to get distracted by your inbox or phone calls
- Help your peers out

 © 2021 General Assembly

Test Driven Development

Test Driven Development

When it comes to building software, as we know, edge cases can often cause bugs. Updating code and adding more features can introduce new bugs and in some cases, break older features in the code. These are just some of the reasons that when building applications, we should also write tests so we can be sure that the application is stable before it reaches production.

Let's be honest, developers don't like writing tests. It feels laborious to write an application and then to go back to the beginning and write hundreds, or in large applications, thousands of tests to make sure that the code we just wrote works when we can just run the app, click around and see for ourselves.

But let's think about this a little more...

Test Driven Development

If you built an app with hundreds of features and thousands of possible user journeys - do you think you would be confident enough in your code to be comfortable saying that it's bug free? What if you release your application and the thought process of the first user is different to yours so they click buttons in a different order and suddenly something is undefined so your code breaks? You fix the bug and at the same time release a new feature. Someone uses the feature and then another part of the app breaks as a result, because you only tested the new feature instead of doing a full regression test.

Tests seem laborious and time consuming to write, but in the long run they save us a lot of time, as we can just run the tests that we previously wrote to make sure that everything is still working as expected.

TDD

TDD stands for Test Driven Development. When working in a TDD workflow, you write your tests *before* you write your code. This means that you know what the benchmark for your code is and what it has to do before you start your project. When working with TDD, you'll start off with a list of tests that all fail (because there's no code yet) and the task of the developer is to write the application in a way that makes all of the tests pass.

Jest

Jest is a javascript testing library. It provides functions to *assert* if a condition is true of false. True means the test passes, false means it fails. Simples.

Using this method, we can run the functions from our application and ask Jest if the condition resulting from the function is the same as what we would expect (we have to provide the correct answer so jest can compare the two).

Guided Walk-Through:
Jest

Follow along at this repo:

week5_lesson2/01-test-driven-development-with-jest.md#jest

Group Exercise:

Lab Time!

In your breakout rooms

Write functions to make the rest of the tests pass. You'll need to:

1. Write the functions from scratch, making sure the function names match the names in the tests.
2. Make sure you add your functions to the module.exports at the bottom of the `functions.js` file
3. Make sure you add them to the imports at the top of the `functions.test.js` file

Copy these tests into your `functions.test.js` file and unskip them as you go

Wrap-ups

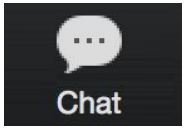
Solo Exercise:
Finish That Sentence

2 mins

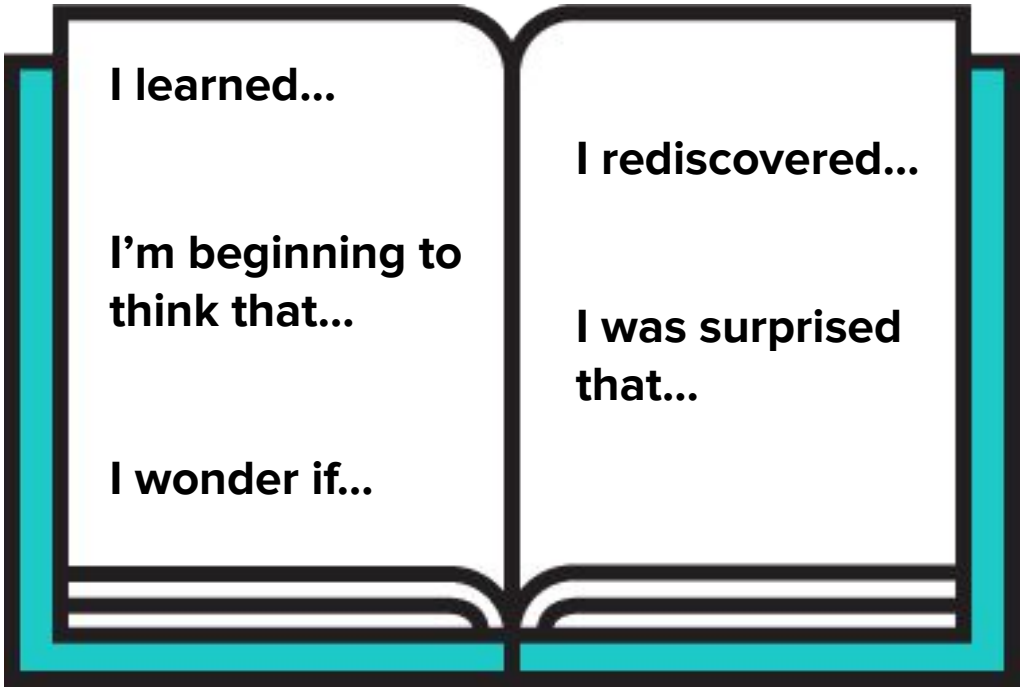
In 30 seconds, we are going to ask you to complete and post **TWO** of these phrases in the **Chat Box**

.

Example:



I learned how to...
I was surprised that...



Coming up!

Unit	Summary
Lesson 1 **Unit 1: Command Line Interface with Linux & Bash** **Distributed Version Control with Git & GitHub**	Practice working with command line and reinforce Unix Shell fundamentals Work with both local and remote Git repositories and with features that enable team collaboration
Lesson 2 **Cloud Providers (3 hours)**	Deep dive into moving to cloud.
Lesson 3 **Cloud Infrastructure (3 hours)**	Understand the benefits of adopting a cloud-based infrastructure and the different models of delivering cloud services.
Lesson 4 - 6 **Containers with Docker and Kubernetes (9 hours)**	Use containers and understand how they fit into modern development workflows.

Unit	Summary
Lesson 7 **Microservices Architecture (3 hours)**	Explain the benefits of microservice architecture and build scalable microservice applications.
Lesson 8 & 9 **Continuous Integration with Jenkins (6 hours)**	Automate your testing and deployments by using continuous integration pipelines in Jenkins.
Lesson 10 **Extreme Programming (3 hours)**	Use XP workflows to manage development workflows more efficiently.
Lesson 11 **Test Driven Development (3 hours)**	Increase test coverage for your code to build and maintain confidence with unit and integration tests.
Lesson 12 **Capstone Projects (3 hours)**	Completed throughout the delivery with a presentation on the last session.