

Developing Salesforce Lightning Web Components (SP-DEX602)



Agenda



- **Day 1**
 - Agenda & Course Overview
 - Introduction to GenWatt
 - Setting up a Practice Site
 - Salesforce DX
 - Scratch Orgs
 - Setting up a Salesforce Developer Hub Org
 - Salesforce CLI
 - Working with Visual Studio Code
 - Installation and Extensions
 - Projects
 - Pushing & Pulling Metadata
 - Model - View - Controller
 - Lightning Architecture
 - Lightning Experience Overview
 - App Launcher
 - Lightning Apps
 - Deploying Code to Production
 - Lightning URLs
 - My Domain
 - Lightning App Builder
 - Lightning Pages
 - Lightning Tabs
 - Lightning Components
 - Standard Components
 - Component Library
 - AppExchange Components
 - Lightning Component Framework
 - Aura Framework
 - Lightning Aura Components Overview
 - Lightning Web Components

Agenda



- **Day 2**
 - Lightning Pages
 - Record Pages
 - Assigning Pages
 - Buttons & Actions
 - Global Actions
 - Lightning Web Components Playground
 - Camel & Kebab
 - Building Your First Components
 - Cascading Style Sheets (CSS)
 - Syntax
 - Selectors
 - Common CSS elements
 - Classes & Subclasses
 - Using CSS in Components
 - Classes
 - Default Styles
 - Design Challenge
 - Salesforce Lightning Design System (SLDS)
 - Building a Tile Component using (SLDS)
 - CSS Tips for Components

Agenda



- **Day 3**
 - Moving from Playground to Scratch
 - Lightning Component Apps
 - Using Lightning Web Components in Aura
 - Hello World App
 - Component Configuration Files
 - Lightning Pages & Tabs
 - Importing External Style Sheets
 - Public Properties
 - Lightning Data Service
 - lightning-record-form
 - Walkthrough: lightning-record-form
 - Nesting Components
 - Events and Event Handling
 - Walkthrough: Events / Conditional Display

Agenda



- **Day 4**
 - Walkthrough: lighting-record-view-form
 - Walkthrough: lighting-record-edit-form
 - SOQL
 - Querying Relationships
 - Apex
 - When to use Apex
 - Key Concepts
 - Collections and Loops
 - Retrieving Data
 - DML
 - Classes
 - Considerations for Lightning Web Components
 - Walkthrough: Apex Controller

Agenda



- **Day 5**
 - Building Complex Components
 - Opportunity Card Component
 - Walkthrough: Opportunity List Controller
 - Review and Discussion
 - Survey
 - Questions & Answers

Course Overview



- About this Course
- About the Instructor
- Introductions
- Course Structure
 - Introduce concept & technology
 - Business Scenario
 - Exercise to use concept in system
- Housekeeping

Business Scenario: GenWatt



GenWatt currently sells generators to large companies and has been using Salesforce Classic for six months. They are growing rapidly. They would like to upgrade from Salesforce Classic to Salesforce Lightning Experience and have hired you to help them.

GenWatt wants to take advantage of the improved ability to customize the user interface that Lightning provides, so they expect to need to build a few Lightning Components.

Salesforce Developer Experience (DX)

- Source-driven Development
- Team Collaboration
- Continuous Integration and Delivery
- Open and Prescriptive Developer Experience
- Rapid Testing and Development

Scratch Orgs



- Source-driven and disposable deployments of Salesforce code and metadata
- Fully configurable
 - Allows developers to emulate different Salesforce editions with different features and preferences
 - Do not have any custom configurations by default
- Do not have data by default
- Available in both Salesforce Classic and Lightning Experience
- Available in **Developer**, **Enterprise**, **Group**, and **Professional** Editions
- Limits
 - 200 MB for data
 - 50 MB for files
- Automatically expires in 7 days

Developer Hub Environment



- The main Salesforce org that you and your team use to create and manage your scratch orgs
- Available in Classic and Lightning
- Available in Developer, Enterprise, Performance, and Unlimited Editions
- Use the Dev Hub to manage scratch orgs
 - Continue using the Environment Hub to manage other types of orgs, including production and trial orgs.

Exercise: Setup Dev Hub Environment



Goal:

Sign up for a Dev Hub Environment.

Action Steps:

1. Navigate to:
<https://developer.salesforce.com/promotions/orgs/dx-signup>
2. Fill out the registration form:
 1. Email: [***your_actual_email_address***]
 2. Username: [***your first name***].[***your last name***]@spdex602.edu
3. Click the Sign Me Up button.
4. Check your email and on the email confirmation message click the Verify Account button.
5. Set your password and security question.
6. Write your username and password on the front page of your book.

Salesforce DX Users



- System administrators can access the Dev Hub org by default
- The following Standard user licenses can be given access to the Dev Hub:
 - Salesforce CRM
 - Salesforce Platform
 - Salesforce Limited Access - Free (partners only)
- To give full access to the Dev Hub org, the permission set must contain these permissions:
 - Object Settings > Scratch Org Info > Read, Create, and Delete
 - Object Settings > Active Scratch Org > Read and Delete
 - Object Settings > Namespace Registry > Read, Create, and Delete
- To work with second-generation packages in the Dev Hub org, the permission set must also contain:
 - System Permissions > Create and Update Second-Generation Packages

Salesforce CLI



- Salesforce Command Line Interface
- Simplifies development and build automation when working with your Salesforce org.
- Capabilities
 - Create development projects
 - Synchronize source to and from:
 - Scratch orgs
 - Sandboxes
 - Production orgs
 - Source control
 - Create and manage orgs
 - Import and export data
 - Create and execute tests
 - Create and install packages

Exercise: Install Salesforce CLI



Goal:

Install the Salesforce Command Line Interface (CLI).

Action Steps:

1. Navigate to: <https://developer.salesforce.com/tools/sfdxcli>
2. Click the Download button.
3. Follow the installation steps.
4. Verify the installation was successful by opening a command prompt.
5. At the prompt, type in **sfdx**.
6. If sfdx is installed, you should see something similar to the text below **Usage: sfdx COMMAND**

Help topics, type **sfdx help TOPIC** for more details:

```
force    tools for the Salesforce developer
help     display help for <%= config.bin %>
plugins  add/remove/create CLI plug-ins
update   update the sfdx CLI
```


Visual Studio Code

- A free code editor from Microsoft
- Features:
 - Intellisense
 - Debug code right from the editor
 - Launch or attach to your running apps
 - Debug with break points, call stacks, and an interactive console
 - Git commands built-in
 - Working with Git and other SCM providers has never been easier. Review diffs, stage files, and make commits right from the editor.
 - Push and pull from any hosted SCM service
 - Extensible
 - Develop extensions to support any language

Visual Studio Code Extensions




- Integrated Development Environment for the Salesforce Lightning Platform
- Built on top of Visual Studio Code
- Major Features
 - Develop Apex Code
 - Develop Visualforce pages and components
 - Develop Lightning Components
 - View and modify Metadata
 - Develop SOQL queries
 - Local copy of all Metadata
 - Can be integrated with source control

Exercise: Install Visual Studio Code

Goal:

Install Visual Studio Code and Extensions.

Action Steps:

1. Navigate to: <https://code.visualstudio.com/>
2. Install Visual Studio Code.
3. Launch Visual Studio Code.
4. On the left toolbar, click the Extensions icon .
().
5. Search for “Salesforce Extension Pack”.
6. Install the extension pack.

Exercise: Install Visual Studio Code (cont...)



Action Steps:

7. Search for “Prettier - Code formatter”.
8. Install the extension pack.
9. Search for “Salesforce Package.xml Generator”.
10. Install the extension pack.

Exercise: Set VS Code Preferences

Goal:

Change the VS Code Color Theme.

Action Steps:

1. Launch Visual Studio Code.
2. In the top menu, click
 - a. Mac: Code -> Preferences -> Color Theme
 - b. Windows: File -> Preferences -> Color Theme
3. Select a Color Theme you like.

Salesforce CLI commands



- Copy a git project
 - `git clone url_path_to_project`
- Authenticate to your Dev Hub
 - `sfdx force:auth:web:login`
 - o `-d` – Default
 - o `-a` – Alias
- Get the latest version of the CLI
 - `sfdx update`
- Display all orgs
 - `sfdx force:org:list`
- Push metadata to a Scratch Org from the local source
 - `sfdx force:source:push`
- Pull metadata from a Scratch Org to the local source
 - `sfdx force:source:pull`
- Pull metadata from a sandbox or production org to local source
 - `sfdx force:source:retrieve`

Steps to Create an Application

1. Set up your project.
2. Authorize the Developer Hub org for the project.
3. Configure your local project.
4. Create a scratch org.
5. Push the source from your project to the scratch org.
6. Develop the app.
7. Pull the source to keep your project and scratch org in sync.
8. Run tests.
9. Add, commit, and push changes. Create a pull request.
10. Deploy your app using one of the following methods:
 - a. Build and release your app with managed packages.
 - b. Build and release your app using the Metadata API.
 - c. Build and release your app by pushing it to production.

Salesforce DX Projects



- Contain metadata
- sfdx-project.json
 - The definition of the project
 - Path, Namespace, API Version
 - Most projects will store the source code in the force-app folder
- .forceignore
 - used to exclude files on synchronization

Exercise: Create a Project

Goal:

Create a new project in VS Code.

Action Steps:

1. Launch VS Code.
2. In the top menu, click View -> Command Palette
3. Type in “SFDX: Create Project with Manifest” and select the command.
4. Choose “Standard” for the project template.
5. Name the project **spdex602** and save the folder in your Workspace folder and click “Create project”.
6. In the top menu, click File -> Save Workspace As... and save the workspace with the name **“spdex602.code-workspace”**.

Exercise: Connect CLI to Dev Hub



Goal:

Connect your Dev Hub environment to Salesforce CLI.

Action Steps:

1. Launch VS Code.
2. In the top menu, click View -> Terminal.
3. In the terminal window, type the following command

```
sfdx force:auth:web:login -d -a  
"DevHub"
```
4. Your browser will launch a Salesforce login screen.
5. Login to your Dev Hub org.
6. You will be prompted to allow access.
7. Click the Allow button.
8. You will see a message in the command window indicating you were successfully authorized.

project-scratch-def.json



- Configuration file that sets the default values for new scratch orgs
- Very flexible with many options
 - edition
 - Sets the edition of the org
 - Developer, Enterprise, Group, Professional
 - adminEmail
 - The email address of the user that gets created
 - hasSampleData
 - If set to true will include some sample data
 - features
 - Over 50 features that can be enabled for the org
 - Examples: CPQ, EinsteinAssistant, Pardot, WavePlatform
 - settings
 - Support changing the options for many of the built-in objects
 - All settings from the Metadata API Settings are available

Exercise: Modify Scratch Definition



Goal:

Modify the project-scratch-def.json file to include your email address and sample data in the scratch orgs you create

Action Steps:

1. Launch VS Code
2. Navigate to the SPDEX602 workspace and spdex602 folder
3. Under the config folder, click on the **project-scratch-def.json** to open the file
4. In the file, add the following entries to the json:

```
"hasSampleData":true,  
  "adminEmail": "[your email address]"
```
5. Save the changes to the file.

Exercise: Create a Scratch Org



Goal:

Create a scratch org to develop your project.

Action Steps:

1. Launch Visual Studio Code.
2. In the top menu, click View -> Terminal.
3. In the terminal window, switch to the **spdex602** directory.
4. Type the following command to create the org

```
sfdx force:org:create
--setdefaultusername
-f config/project-scratch-def.json
--setalias my-scratch-org
```
5. Type the following command to login to the scratch org:

```
sfdx force:org:open
```

Pushing and Pulling MetaData

- The Scratch Org and VS Code are connected, but not automatically synchronized
- Synchronization of code and metadata is performed through the Salesforce CLI
- Push metadata to a Scratch Org from the local source
 - `sfdx force:source:push`
- Pull metadata from a Scratch Org to the local source
 - `sfdx force:source:pull`
- Retrieve metadata from a sandbox or production to the local source
 - `sfdx force:source:retrieve`
- Deploy metadata from local source to sandbox or production
 - `sfdx force:source:deploy`

Exercise: Create an LWC and Push It



Goal:

Create a Lightning Web Component using VS Code and push it to your Scratch Org using the Salesforce CLI.

Action Steps:

1. Launch Visual Studio Code and Open your `spdex602` Workspace.
2. In the `spdex602` project, under the **`force-app\main\default`** directory, right click on the **`lwc`** directory and click SFDX: Create Lightning Web Component.
 - a. Name the component **`lwcExample`**
 - b. Alternative:
 - a. Right click on the **`lwc`** folder and click on "Open in Terminal".
 - b. `sfdx force:lightning:component:create -n lwcExample --type lwc`
 - c. Press enter to confirm the default directory.
3. Open the terminal window.
4. Type the following command to push the `lwcExample` component to your Scratch Org:
 - o `sfdx force:source:push`
5. Login to your Scratch org using `sfdx force:org:open`
6. Navigate to Setup and search for Lightning Components.
7. Verify that the **`lwcExample`** component exists in your Scratch org.

Exercise: Create an Apex Class and Pull It



Goal:

Create an Apex Class using Salesforce and pull it to your local project using Salesforce CLI.

Action Steps:

1. Login to your Scratch org.
2. Navigate to Setup and search for Apex Classes.
3. Create a New class by clicking the new button.
4. Create a class using the following code and save it:

```
public class dxSample {}
```
5. Launch Visual Studio Code and Open your Salesforce Workspace.
6. Open the terminal window.
7. Type the following command to pull the dxSample2 class to your local project: `sfdx force:source:pull`
8. Verify the **dxSample** class has been added to the project.

Technical Scenario: Production Org



To simulate how a developer works in an enterprise environment, we are going to be using our DevHub organization as our production environment. During this class, we will develop our code in the scratch org and perform testing.

Note: In an actual enterprise environment, we would deploy to Sandbox to test, but we do not have sandboxes available.

Steps for Syncing Scratch with Production



1. Authorize production by logging in through the CLI.
2. Pull the code from the scratch org to local.
3. Deploy the code from local to production.
4. Pull the code from production to local.
5. Push the code from local to scratch.

Manifest



- The manifest folder contains the package.xml file
- package.xml
 - The definition of the metadata to sync with this project to and from sandbox and production orgs
 - <types>
 - A container for the type of metadata to sync
 - <name>
 - A child of <types>
 - The specific type of metadata to sync
 - <members>
 - A child of <types>
 - The specific metadata files to sync
 - Can use * to sync everything
 - Also used by the Force.com IDE and Metadata API developers

Exercise: Deploy Production Metadata



Goal:

Deploy metadata from your local project and merge it with your production org (Dev Hub).

Action Steps:

1. In VS Code, right click on the spdex602 folder and click Open in Terminal.
2. Deploy the metadata from your local project by using the following command:

```
sfdx force:source:deploy -u [your  
username for production] --manifest  
"[full path to your package.xml]"
```
3. Verify that the dxSample and lwcExample files were deployed.

Exercise: Retrieve Production Metadata



Goal:

Retrieve metadata from your production org and merge it with your local project.

Action Steps:

1. In VS Code, right click on the spdex602 folder and click Open in Terminal.
2. Retrieve the metadata from your production org by using the following command:

```
sfdx force:source:retrieve -u [your  
username for production] --manifest  
"[full path to your package.xml]"
```

3. Verify that three Getting Started files were downloaded and stored in the force-app/main/default/aura directory of your spdex602 project.

Exercise: Synchronize with Scratch Org



Goal:

Push the metadata from your local project to your scratch org to ensure your scratch org and production are in sync.

Action Steps:

1. In VS Code, right click on the spdex602 folder and click Open in Terminal.
2. Push your code to scratch using the following command:

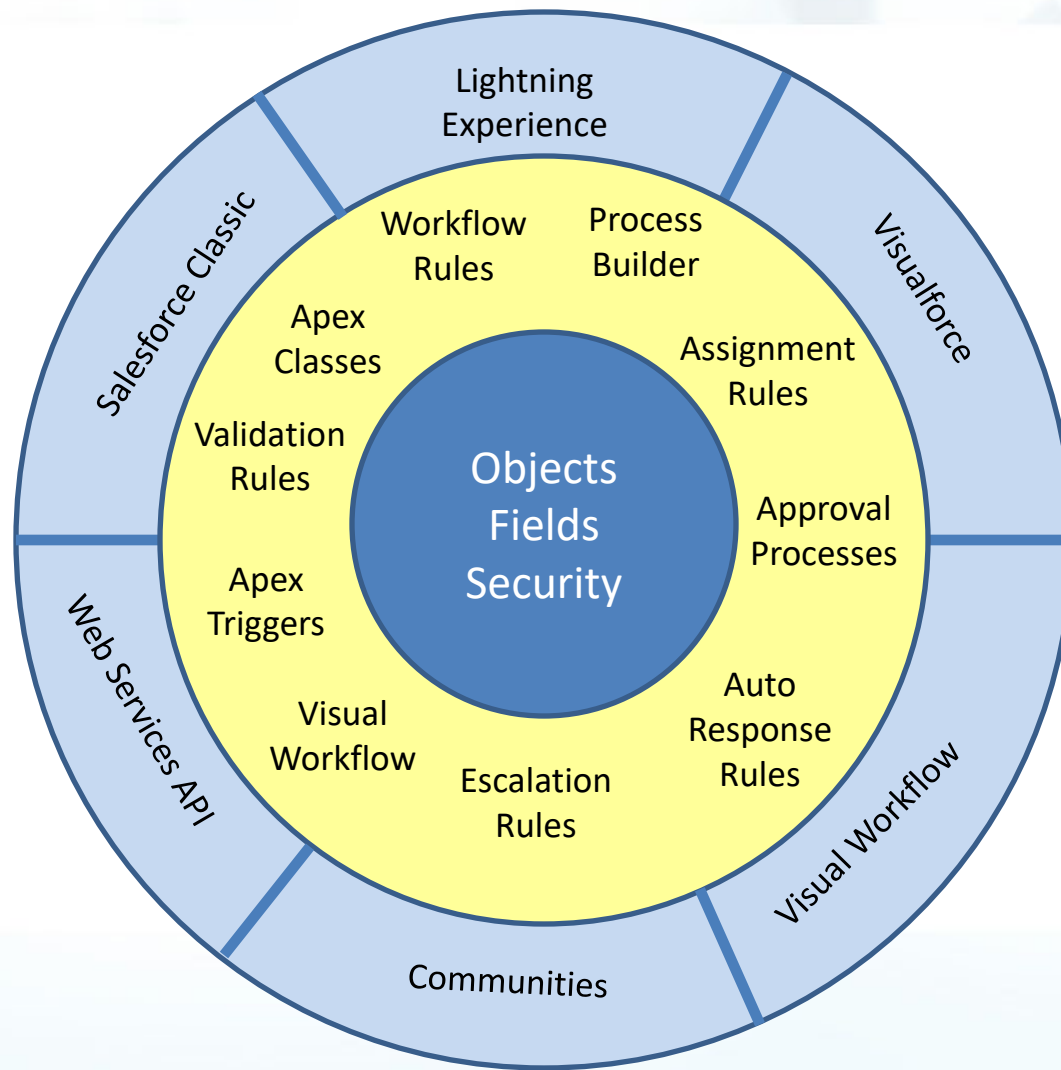
```
sfdx force:source:push
```
3. Verify that only the three Getting Started files were synchronized.
4. Pull your code from scratch using the following command:

```
sfdx force:source:pull
```
5. Verify that there were no changes.
 - a. When working with a scratch org, only files that have changed will synchronize

Model – View – Controller (MVC)

- The idea that software development can be separated into three distinct functional areas
- Academic in nature and not practical in reality
- Salesforce attempts to support this paradigm
- **Model**
 - The database layer
 - Contains objects, fields, and security
- **Controller**
 - The business logic layer
 - Links the Model to the View
 - Contains Workflow, Validation Rules, Apex
- **View**
 - The user interface layer
 - Tabs, Page Layouts, Visualforce

Model - View - Controller





Lightning Platform Architecture

- Salesforce separates its architecture into two distinct development methods: Declarative and Programmatic
- Declarative
 - Clicks not code
 - The preferred method of development
 - Automatically upgraded
 - Examples: Page Layouts, Validation Rules, Workflow Rules, Formula Fields
- Programmatic
 - Code
 - Apex / Visualforce
 - Lightning Components
 - Web Services API using standard tools
 - Not automatically upgraded
 - Locked to a specific version of the API
 - Should only be used as a last resort

Lightning Platform Architecture & MVC



	Declarative	Programmatic
View	Page Layouts Tabs List Views Visual Workflow Reports & Dashboards	Visualforce Lightning Components External Program
Controller	Validation Rules Workflow Rules Process Builder Visual Workflow Approval Processes Assignment Rules Auto-Response Rules Escalation Rules	Apex Triggers Apex Classes Web Services API Integration
Model	Objects External Objects Profiles / FLS OWD / Sharing	Objects External Objects Profiles / FLS OWD / Sharing

On the Lightning Platform, the Model & Controller behave as a single unit

Lightning Experience



The screenshot displays the Lightning Experience user interface. At the top, there is a navigation bar with a search bar labeled "Search Opportunities and more...". Below the navigation bar, the main content area shows an Opportunity record for "United Oil & Gas Corp.-Existing Customer - Upgrade-3/2016". The record details include the Account Name "United Oil & Gas Corp.", Close Date "3/19/2016", Amount "\$270,000.00", and Opportunity Owner "Student One". A progress bar at the top of the record shows stages from "Lead" to "Negotiation...", with the current stage highlighted. Below the progress bar, the "DETAILS" tab is active, showing a list of fields and their values. To the right of the details, there are sidebars for "Products (0)", "Contact Roles (0)", and "Stage History (2)". The "Stage History" sidebar shows two stages, both with the same details as the current stage.

Opportunity
United Oil & Gas Corp.-Existing Customer - Upgrade-3/2016

+ Follow Edit New Case New Note

Account Name Close Date Amount Opportunity Owner
[United Oil & Gas Corp.](#) 3/19/2016 \$270,000.00 [Student One](#)

✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ Negotiation... Closed ✓ Mark Stage as Complete

ACTIVITY CHATTER DETAILS

Opportunity Name
United Oil & Gas Corp.-Existing Customer - Upgrade-3/2016

Opportunity Record Type

Account Name
[United Oil & Gas Corp.](#)

Type
Existing Customer - Upgrade

Lead Source

Primary Campaign Source

Description

Opportunity Owner
[Student One](#)

Close Date
3/19/2016

Next Step

Stage
Negotiation/Review

Close Reason

Probability (%)
90%

Amount
\$270,000.00

Negotiated Discount Amount
\$0.00

Discount Rate

Discount Amount

Products (0)

Contact Roles (0)

Stage History (2)

Stage: Negotiation/Review
Amount: \$270,000.00
Probability (%): 90%
Expected Revenue: \$243,000.00
Close Date: 3/19/2016
Last Modified By: [Chris Ude](#)
Last Modified: 3/19/2015 10:20 AM

Stage: Negotiation/Review
Amount: \$270,000.00
Probability (%): 90%
Expected Revenue: \$243,000.00
Close Date: 2/17/2010
Last Modified By: [Chris Ude](#)
Last Modified: 3/16/2015 7:56 AM

[View All](#)

App Launcher



- Replaces the App Menu in Classic
 - Displays Lightning Apps, not Classic Apps
 - Classic Apps can be displayed in Lightning
- Allows users to navigate to both Apps & Tabs
- Users can navigate to the AppExchange
- Apps & Tabs can be searched
- Setup is an App
 - Setup App is not accessible through the App Launcher

Lightning Apps



- More Robust and Functional than Classic Apps
- Branding
 - Custom logo
 - Custom color scheme
- Navigation Style
 - Standard
 - Console
 - Service Setup
- Form Factor
 - Desktop
 - Phone
 - Desktop and Phone
- Utility Bar
- Items
 - Currently the only Items you can add are Tabs

Exercise: Lightning App



Goal:

Create a Lightning App in your Scratch Org.

Action Steps:

1. Download the GenWatt logo for this exercise from <http://materials.stonyp.com/genwattlogo.png> and save it to your desktop.
2. Login to your Scratch Org from VS Code using `sfdx force:org:open`
3. Navigate to Setup and search for “App Manager”.
4. Create a new Lightning App by clicking the *New Lightning App* button.
5. App Details & Branding:
 - a. App Name: **GenWatt Sales**
 - b. Developer Name: **GenWatt_Sales**
 - c. Image: upload the **genwattlogo.png**
 - d. Click the Next button.
6. App Options:
 - a. Navigation Style: **Standard navigation**
 - b. Supported Form Factors: **Desktop**
 - c. Click the Next button.

Exercise: Lightning App (cont ...)



Action Steps:

7. Utility Bar:
 - a. Click the Add Utility Item button and add the Recent Items component.
 - b. In the Component Properties section, Select the following Objects: **Account, Contact, Opportunity.**
 - c. Set the Number of Records to Display to **10.**
 - d. Click the Next button.
8. Select the following Items in the order below:
 - a. **Home, Tasks, Calendar, Accounts, Contacts, Opportunities, Campaigns, Reports, Dashboards, Scorecards.**
 - b. Click the Next button,
9. Assign the App to the following profiles:
 - a. **Custom: Sales Profile, System Administrator.**
 - b. Click the Save & Finish button.
10. Using the App Launcher, navigate to the GenWatt Sales app.

Exercise: Set Scratch Org Login



Goal:

Set a user and password for your scratch org that will allow you to login directly to the scratch org without having to use VS Code.

Action Steps:

1. In VS Code, run the `sfdx force:org:open` to open your Scratch org.
2. Navigate to Setup and search for “Users”.
3. Navigate to the Users page and edit the User, User record and set the following values:
 - a. First Name: **[your first name]**
 - b. Last Name: **[your last name]**
 - c. Username: **[first].[last]@spdex602.org**
 - d. Email: **[your email address]** (if it’s not set already)
4. Save the user record.
5. Check the box next to your User record and click the Reset Password(s) button.
6. Check your email and follow the steps to reset your password.

Exercise: Update package.xml



Goal:

Update the package.xml file to include the metadata that Salesforce created when you built the GenWatt Sales App.

Action Steps:

1. From the command palette in VS Code, run **SFDX Package.xml Generator: Choose Metadata Components** command.
2. Select “**ContentAsset**”, “**CustomApplication**” and “**FlexiPage**”, then click “**Update Package.xml**”.
3. In the **package.xml** file, change the **ContentAsset** <members> from “*” to “genwattlogo”, the **CustomApplication** <members> from “*” to “GenWatt_Sales” and the **FlexiPage** <members> from “*” to “GenWatt_Sales_UtilityBar”. It should look like this:

```
<types>
  <members>genwattlogo</members>
  <name>ContentAsset</name>
</types>
<types>
  <members>GenWatt_Sales</members>
  <name>CustomApplication</name>
</types>
<types>
  <members>GenWatt_Sales_UtilityBar</members>
  <name>FlexiPage</name>
</types>
```

4. Save the **package.xml** file.

Exercise: Deploy to Production



Goal:

Deploy the GenWatt Sales App from local and push it to production.

Action Steps:

1. In VS Code, right click on the spdex602 folder and click Open in Terminal.
2. Pull your code from scratch using the following command:

```
sfdx force:source:pull
```
3. You should get a number of files when you pull the source, including:
 - a. A FlexiPage, a CustomApplication, a Profile and an AppMenu
4. Deploy your code to production using the following command

```
sfdx force:source:deploy -u [your username for production] --manifest "[full path to your package.xml]"
```
5. If everything worked correctly, your GenWatt Sales App will be deployed to your production environment.

Exercise: Test in Production



Goal:

Login to your production org, assign the GenWatt Sales App to the System Administrator profile and verify that the logo displays properly.

Action Steps:

1. Login to your production environment.
2. Navigate to Setup and search for “Profiles”.
3. Edit the System Administrator Profile.
4. In the Custom App Settings section, check the Visible box for the GenWatt Sales (GenWatt_Sales) app and check the box to make it the default.
5. Save your changes.
6. Refresh your page.
7. Verify that the GenWatt Sales App is visible in your App Launcher and displays the GenWatt logo.

Lightning URLs

- Universal Resource Locator
 - The path in the browser bar
 - Classic Example: <https://na50.salesforce.com/>
 - Lightning Example:
<https://na50.lightning.force.com/>
- Impacted in Lightning
 - My Domain
 - URL Hacks
 - Buttons
 - Links
 - Report Parameters

My Domain



- A subdomain within a Salesforce org
- Advantages
 - Supports single sign-on
 - Custom login page
 - Use Salesforce as an identify provider
 - Consistent URLs in future org splits or migrations
- Requirement to build and install custom Lightning Components
- Maximum of 40 characters


Exercise: Configure My Domain



Goal:

Configure the login screen for My Domain in your scratch org.

Action Steps:

1. In VS Code, run the `sfdx force:org:open` to open your Scratch org.
2. Navigate to Setup and search for “My Domain”.
3. In the Authentication Configuration section, click the  button.
4. Set the Header Logo by uploading the **genwattlogo.png** file.
5. Click on the Save button then logout of Salesforce.
6. When you logout, you should be redirected to your custom login page and see the GenWatt logo.
 - a. The URL will not be login.salesforce.com but will instead be the full, custom domain you created.

Lightning App Builder



- Used to create custom Lightning Pages for Lightning Experience and the mobile app (formerly called Salesforce1)
- Can build:
 - App Pages
 - Good for Dashboards & Mobile Apps
 - Not bound to a specific record or records
 - Home Pages
 - Can only be used on the Home Page in Lightning Experience
 - Not available on Mobile
 - Not bound to a specific record
 - Record Pages
 - Bound to a specific record
 - Email Application Pane
 - For use with Outlook Integration
 - Embedded Service Page
 - For use with Lightning Field Service
 - Lightning Apps
- Templates
 - There are 11 standard templates available to control the layout of the page
- All Lightning Pages are visible & editable from within the App Builder

Lightning Pages



- A more modern User Experience
- Consist of one or more Page Regions
 - Defined by the Page Template
 - Contain Lightning Components
 - Components available vary by the type of page being built
- Don't replace Page Layouts
 - Overlay existing Page Layouts
- Must be Activated to use
- Should be Assigned to Users
- Lightning Record Pages don't truly exist until you edit the Page and save changes
 - There is a default, system generated page that cannot be customized and will be the default page
 - As soon as you modify the page, a Lightning Page is created
 - Each Object can have multiple Lightning Record Pages

Lightning Components



- Two Categories
 - Standard
 - Custom
- Four types
 - Lightning Aura Components
 - Standard
 - Custom
 - Lightning Web Components
 - Standard
 - Custom
- The building blocks of the Lightning Experience
- Provide all the functionality on a Lightning Page
- Can have attributes (properties) that give the component flexibility
- Some components can contain other components
 - Example: Tabs Component
- Can be set to show / hide based on values on the page
- Developers can create Custom Components

Standard Lightning Components



- Provided by Salesforce out of the box
 - Lightning Web Components
 - Aura Components
- Cannot be modified
 - Most have attributes that allow configuration
- Cannot be deleted

Component Library



- A library of all Lightning Components available in your org
 - Path:
<https://<myDomain>.lightning.force.com/docs/component-library>
 - This library will include any custom components you develop or install
- You can also use the Component Library without logging in to Salesforce.
 - Path:
<https://developer.salesforce.com/docs/component-library>
 - Will only include standard Lightning Components

AppExchange Components



- Salesforce added a new section to the AppExchange to allow developers to share Custom Components
 - Change the Listing by Type to find Components instead of Apps
- Can be free or paid
- Approximately 200 components available now

Lightning Component Use Cases

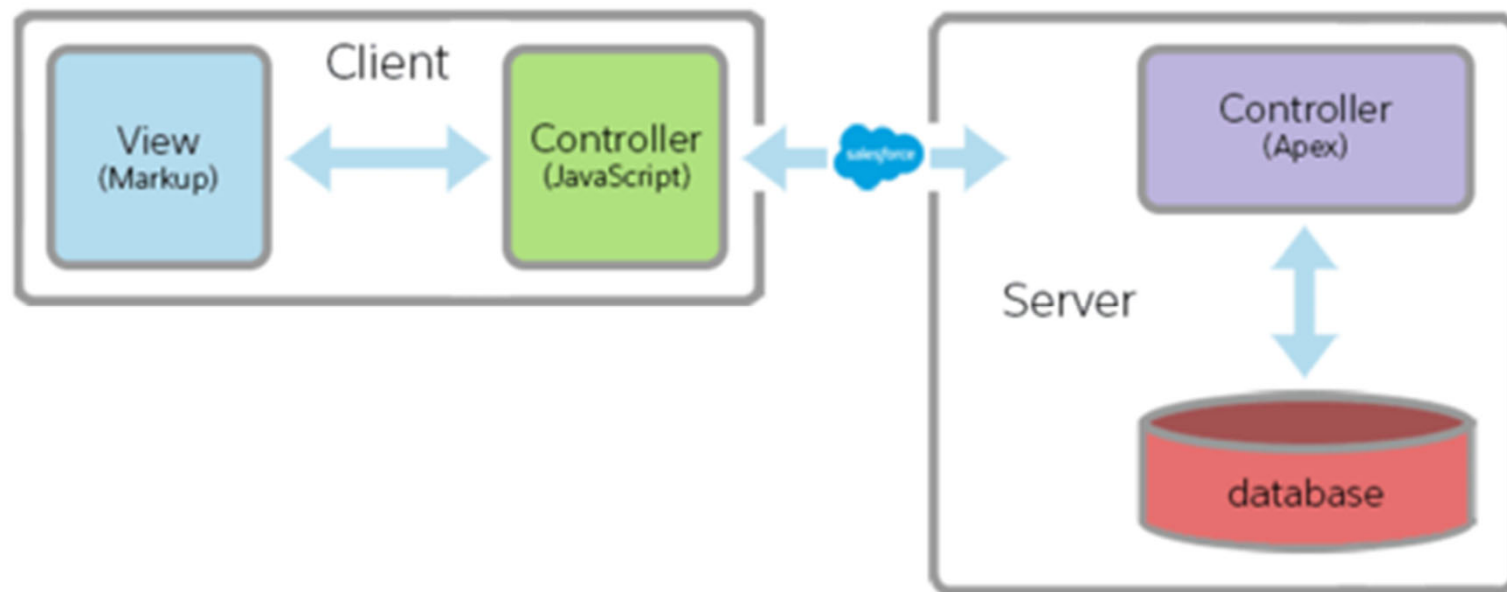


- Lightning Experience
 - Tabs
 - Pages
 - Record Pages
 - Actions
 - Button Overrides
- Mobile
- Community
- Flows
- Chatter
- Visualforce Pages
- Lightning Out (beta)

Lightning Component Framework

- A User Interface (UI) framework for developing apps for:
 - Phones
 - Tablets
 - Desktops
- Mobile development is typically a single-page application
- Consists of:
 - A client-server framework
 - XHTML, JavaScript, CSS
 - A server-side framework
 - Apex
 - Lightning Platform

Lightning Component Framework



Lightning Web Components (LWC)



- The replacement for Lightning Aura Components
- Uses core Web Components (w3c) standards instead of a unique framework
 - Performance is better because the framework doesn't need to be downloaded and updated
- Consist of 3 file types on the client
 - HTML
 - JavaScript
 - CSS
- Interact with Salesforce data by calling server-side methods
 - Lightning Data Service
 - Apex methods
 - API Calls
- Work primarily by interacting with Events

LWC vs Aura Components

Resource	Aura File	Lightning Web Components File
Markup	sample.cmp	sample.html
Controller	sampleController.js	sample.js
Helper	sampleHelper.js	
Renderer	sampleRenderer.js	
CSS	sample.css	sample.css
Documentation	sample.auradoc	Not currently available
Design	sample.design	sample.js-meta.xml
SVG	sample.svg	Included in HTML file or upload as a static resource

- LWCs can be included in Aura, but Aura cannot be included in LWCs.
- LWCs offer better performance because there is no added abstraction layer such as the one with Aura components.
- The Aura Framework is not going away, but future development should be done with Lightning Web Components.

- Allows developers to contribute to the roadmap
- Learn the framework at a deeper level by exploring the source code
- Use the same framework whether building apps on Salesforce or any other platform.

Supports Lightning Web Components



- Lightning Experience
- Salesforce App
- Lightning Communities
- Lightning App Builder
- Community Builder
- Standalone Apps
- Lightning Components for Visualforce
- Lightning Out (beta)
- Custom Tabs
- Utility Bars
- Flows
- First-Generation Managed Packages
- Second-Generation Managed Packages
- Unlocked Packages
- Unmanaged Packages
- Change Sets
- Metadata API—LightningComponentBundle
- Tooling API—LightningComponentBundle, LightningComponentResource
- EMP API
- Embedded Service Chat
- Gmail and Outlook integration

No Support for Lightning Web Components

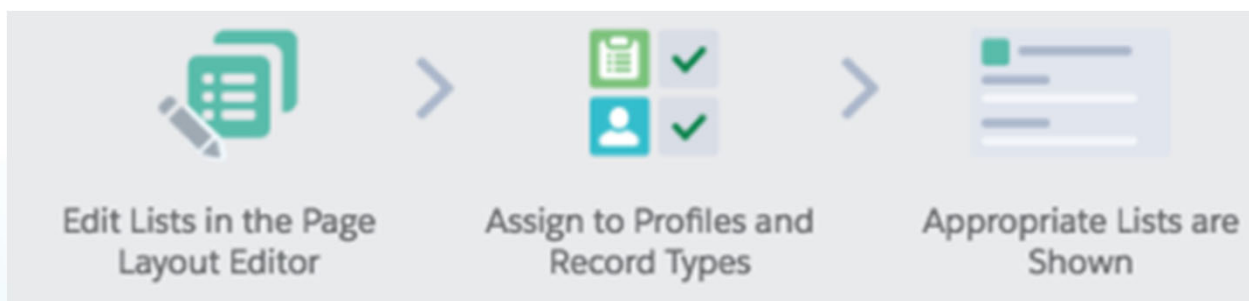


- Lightning Web Components doesn't currently support these Salesforce experiences and tools:
 - Salesforce Console (Navigation Item API, Workspace API, UtilityBar API)
 - URL Addressable Tabs
 - Conversation Toolkit API, Omni Toolkit API, Quick Action API
 - Standard Action Overrides, Custom Actions, Global Actions, List View Actions, Related List View Actions
 - Chatter Extensions
- To use a Lightning web component with these experiences and tools, wrap the component in an Aura component

Page Layouts in Lightning Experience



- Behave as the data source for several Lightning Components
 - Record Detail
 - Fields display in the same order on the Lightning Record Page as they do on the page layout
 - Related Lists
 - The related lists on the page layout display in the same order on the Lightning Record Page
 - Highlights Panel
 - The Lightning Actions display in on the Lightning Record in the same order they display on the page layout



Editing Lightning Pages



- With the Lightning App Builder
 - Most Components
- With the Classic Page Layout Editor
 - Detail Section
 - Related Lists
 - The ones displayed
 - The order of the columns displayed
 - Maximum of 5 columns will display in a related list Lightning
 - The first 5 columns will display
 - If the User clicks View All, all 10 columns can display
 - Lightning Actions displayed
 - Actions in the Salesforce Mobile and Lightning Experience Actions will display in the Lightning Experience
 - They will display in order
 - Some Actions will display in the Page Layout Editor but will not actually display on the Page due to feature and security settings
 - Similar behavior to Classic

Exercise: Lightning Account Record Page



Goal:

Create a Lightning Record Page in your scratch org for the Account object.

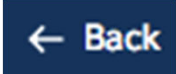
Action Steps:

1. Navigate to Setup -> Object Manager -> Account -> Lightning Record Pages
2. Click the New button.
3. Choose to create a Record Page and click the Next button.
 - a. Label: **GenWatt Account**
 - b. Object: **Account**
4. On the Choose Page Template screen, choose the “Header and Two Equal Regions” template and Click the Finish Button.
5. Drag the **Highlights Panel** component to the Header region.
6. Drag the **Tabs** component to the region on the left of the page.
7. You will see Tabs appear in the property pane on the far right.
 - a. Click on the Related tab and change the Tab Label to **Chatter**.
 - b. Reorder the tabs so that they display as **Chatter, Details**.
 - c. Set the Default Tab to **Details**.

Exercise: Account Record Page (continued)

8. Click on the Chatter tab and drag the **Chatter** component onto the Chatter tab of the Tab component.
9. Click on the Details tab and drag the **Record Detail** component onto the Details tab of the Tab component.
10. Drag another **Tabs** component to the region on the right of the page.
11. You will see Tabs appear in the property pane on the far right.
 - a. Click on the Details tab and change the Tab Label to **Custom**.
 - i. Enter **Opportunities** as the Custom Tab Label.
 - b. Reorder the tabs so that they display as **Relate, Opportunities**.
 - c. Set the Default Tab to **Related**.
12. Click on the Related tab and drag the **Related Lists** component onto the Related tab of the Tab component.
 - a. Set the following in the properties pane:
 - i. Related List Type: **Tile**

Exercise: Account Record Page (continued)

13. Click on the Opportunities tab and drag the **Related List – Single** component onto the Opportunities tab of the Tab component.
 - a. Set the following in the properties pane:
 - i. Parent Record: **Use This Account**
 - ii. Related List: **Opportunities**
 - iii. Related List Type: **Enhanced**
 - iv. Number of Records to Display: **20**
 - v. Show list view action bar: **Checked**
14. Click the Save button.
15. When prompted to Activate the page, choose **Not Yet**.
16. Click the  **Back** button to exit the Lightning App Builder.

Assigning Apps & Pages



- Lightning Apps are made available to Users at the Profile level using the same rules as Salesforce Classic
- Lightning Pages are assigned using Activation
- Home Page
 - Assigned using Profiles
- App Page
 - Assigned by granting permission to see the Lightning Tab at the Profile level
- Lightning Record Page
 - The most complex assignment
 - Three options:
 - Org Default
 - App Default
 - App, Record Type, and Profile

Assigning Lightning Record Pages

- Org Default (recommended)
 - The Lightning Page every user sees for any record of that object
- App Default (optional)
 - The Lightning Page every user sees for any record of that object when that object is viewed within that specific app
 - This introduces the possibility that the same user will see the exact same record differently depending from which Lightning App they are viewing the record
- App, Record Type, and Profile (recommended)
 - The Lightning Page that will be displayed for a particular user based on the App they are using, their profile and the record type of the record they are viewing
 - You must use this method if:
 - You want to display the same interface to the same user every time they view a record
 - AND you are using Record Types
 - AND you are using multiple Page Layouts
- **Best Practice:** Create one Lightning Record Page per Page Layout

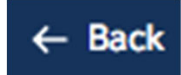
Exercise: Assigning Lightning Pages



Goal:

Assign the Account Record Page as the default page for all users.

Action Steps:

1. Navigate to Setup -> Object Manager -> Account -> Lightning Record Pages.
2. Click the GenWatt Account Record Page.
3. Click the Edit button.
4. In the Lightning App Builder, click the Activation... button in the top right-hand corner of the page.
5. Choose to Assign as Org Default.
 - a. Choose Desktop at the form factor.
 - b. Click the Next button and then click the Save button.
6. Click the  button to exit the Lightning App Builder.
7. Using the App Launcher, Navigate to the GenWatt Sales app.
8. Find the Account record for Edge Communications and verify that the tabs have been properly reordered and the Opportunities sub-tab displays all the Edge Communications Opportunity records.

Challenge: Test in Production



- Use Package.xml Generator to login to scratch and update package.xml
- Pull the source from scratch to local
- Deploy the source from local to production

Lightning Tabs



- Can be created and edited from the Tabs screen in Setup
- Lightning Page Tabs
 - Display Lightning App Pages
 - Cannot be used to display Record Pages or Home Pages
 - Can be created while building an App Page
- Lightning Component Tabs
 - Display Lightning Aura Components or Lightning Web Component
 - Only display in:
 - Navigation Menu in Lightning Experience
 - Mobile app

Buttons

- Lightning doesn't use Buttons
- Lightning uses Actions
- Most standard and custom buttons are automatically available as Actions
- Buttons that might not work properly in Lightning
 - URL Buttons
 - Visualforce Buttons
- Buttons that will not work in Lightning
 - JavaScript buttons

Actions



- Enable users to do more in Salesforce and in the Salesforce App
- Categories
 - Standard Chatter actions
 - Nonstandard actions
 - Default actions
 - Mobile smart actions
 - Custom actions
 - Productivity actions
- Capabilities
 - Create a record
 - Send an email
 - Log a call
 - Display a Visualforce page
 - Display a Canvas App
 - **Display a Lightning Component**

Actions

- Three Types Action
 - Global Actions
 - Object Specific Actions
 - Custom Canvas Actions
- Actions can be added to:
 - Record detail pages
 - Cannot add Chatter Publisher Actions to the Chatter component on Lightning Record Pages
 - Global Publisher
 - Chatter publisher on the home page
 - Chatter tab
 - Chatter groups

Lightning Web Components Playground



- A component of the Lightning Component Library
- An interactive editor for creating basic Lightning Web Components
 - <https://developer.salesforce.com/docs/component-library/tools/playground> or
 - [https://\[your url\]/docs/component-library/tools/playground](https://[your url]/docs/component-library/tools/playground)
- Apps and Lightning Web Components in Playground are publicly accessible
 - Allows easy sharing of code
 - Allows developers to download and upload projects as .zip files
- Limitations of Playground
 - Doesn't use Lightning Locker Service
 - Can't access Salesforce data
 - Standard Lightning Web Components that import Salesforce JavaScript libraries may not work
 - Must use the ISO date format for date components

Camel & Kebab Case

- Camel Case
 - a typographical convention in which an initial capital is used for the first letter of a word forming the second element of a closed compound
 - Examples: *PayPal*, *iPhone*, *MasterCard*
- Kebab Case
 - A typographical convention in which hyphens are used instead of spaces for identifying elements
 - Examples: *pay-pal*, *i-phone*, *master-card*
- Lightning Web Components expect the following:
 - File names to be written in Camel Case
 - Lightning Web Components to be referenced in Kebab Case
 - Lightning will do the conversion automatically for you
 - Example: A component named `helloWorld.html` will be referenced as `<c-hello-world>` when embedded into a container

Building Your First Component



- The Playground project must have a root component named app
 - The app is actually a Lightning Web Component
- You can build as many additional Lightning Web Components as you want within a project
- The HTML of each Lightning Web Component must begin with the `<template>` tag and end with the `</template>` tag

Exercise: Hello World App

Goal:

Create your first App using the Playground.

Action Steps:

1. Navigate to <https://developer.salesforce.com/docs/component-library/tools/playground>
2. Click the New button to create a new project.
3. Change the Name of the Project to **Hello World App**.
4. Check the Live compilation box.
5. Open the app.html file and update the code so it looks like the code below:

```
1  <template>
2  <div>
3    <h4>Hello World App</h4>
4  </div>
5  </template>
```



Exercise: Hello World Component



Goal:

Create your first Lightning Component using the Playground.

Action Steps:

1. On the Project Files banner, click the  button to create a new folder.
 - a. Name the folder **helloWorld**
2. On the helloWorld folder, click the  button to create a new file.
 - a. Name the file **helloWorld.html**
 - b. Create another file and name it **helloWorld.css**
 - c. Create another file and name it **helloWorld.js**
3. Add the following code to the helloWorld.html file:

```
1 <template>
2 <div>Hello World!</div>
3 </template>
```

4. Add the following code to the helloWorld.js file:

```
1 import { LightningElement, track, api } from 'lwc';
2
3 export default class HelloWorld extends LightningElement {}
```



Exercise: Hello World Component

Goal:

Add your Hello World component to the App.

Action Steps:

1. Update the app.html file so the code looks like the code below:

```
1  <template>
2  <div>
3    <h4>Hello World App</h4>
4    <c-hello-world></c-hello-world>
5  </div>
6  </template>
```

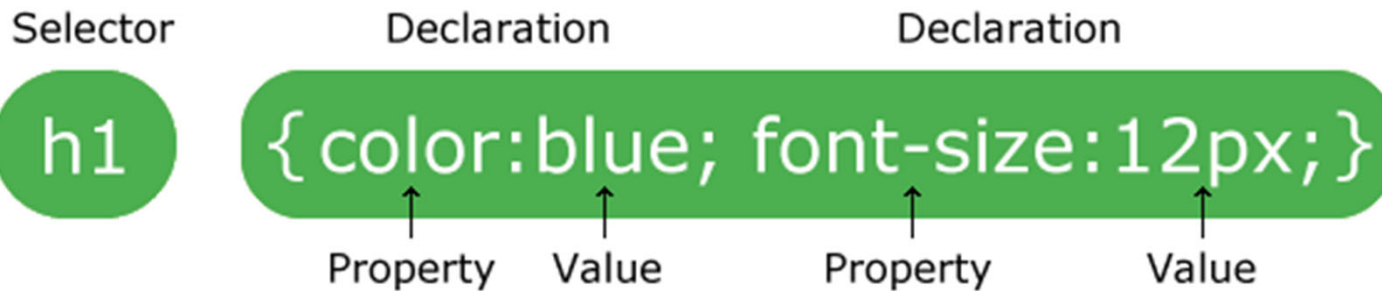
Cascading Style Sheets (CSS)

- A language that describes the style of an HTML document
- Describes how HTML elements are to be displayed
- Styles applied to an element are also applied to elements within that element
 - The styles cascade down, like a stream flowing over rocks
- Consists of selector and declaration blocks
- Selectors point to the HTML element(s) to be styled
 - All elements meeting the selector criteria will be styled the same way
- Declaration block contains one or more style declarations separated by semicolon(s)

CSS in LWC

- Each component can have its own stylesheet
- To create a stylesheet for a component, add a `[componentname].css` file to the component's folder
 - If the component is named `helloWorld`, add a `helloWorld.css` to the component folder
- The style sheet is available to the component automatically
- The style is restricted to the component only and will not flow to parents
 - This fixes a problem that exists in Aura

CSS Syntax



- Example:

```
p {  
  color: red;  
  text-align: center;  
}
```

- All paragraphs will be aligned center and their text will be red

CSS Selectors

- Find HTML elements to apply styles to
- Can find elements based on type, name, id, class, attribute, etc.
- Element selector
 - Based on HTML tags such as `<p>`, `<h1>`, `<body>`, etc.
 - Not recommended in Lightning Components
- Id selector
 - Each element on a page should have a unique Id, so the Id selector only applies to one element
 - To specify an Id selector, begin the style with a `#`
 - The following style rule would be applied to only the element with the `id="helloWorld"`

```
#helloWorld {text-align: center;}
```
 - Id selectors not allowed in LWC.

CSS Selectors

- Class selector
 - Select elements with a specific class attribute
 - To specify a class selector, begin the style with a .
 - The following style rule would apply to any HTML element with the attribute class="bodyText"

```
.bodyText {text-align: left;}
```

- Limiting class selector scope
 - The following style rule would only apply to <p> elements with the class="bodyText" attribute

```
p.bodyText {text-align: left;}
```

- Grouping Selectors
 - If the same style should apply to multiple elements, the elements can be grouped as follows:

```
h1, h2, h3 {text-align: center;}
```

Common CSS elements

- Text
- Fonts
- Colors
- Borders
- Margins

Text

- `color`
 - Set by name, HEX value or RGB value
- `text-align`
 - center, left, right, justify
- `text-decoration`
 - none, overline, line-through, underline
- `text-transform`
 - uppercase, lowercase, capitalize
- `text-indent`
 - Using pixels: `text-indent: 50px;`
- `letter-spacing`
 - Using pixels: `letter-spacing: 2px;`
- `line-height`
 - Example: `line-height: 2.0;`

Fonts

- font-family
- font-style
 - normal, italic, oblique
- font-size
 - px – Specify the exact size in pixels
 - em – Relative to the screen size, allows the user to resize text using the browser
 - vw – Responsive text that will resize as the window resizes
- font-weight
 - normal, bold

Example:

```
1  h4 {
2      font-size: xx-large;
3      font-family: Arial, Helvetica, sans-serif;
4      font-weight: bolder;
5  }
6  div {
7      font-family: Arial, Helvetica, sans-serif;
8  }
```

Colors

- Can be specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values
 - CSS supports 140 named colors
 - https://www.w3schools.com/colors/colors_names.asp
- background-color
 - sets the background color of the HTML element
- color
 - sets the text color for the text within an HTML element
- border
 - Has attributes that allow the color of the border to be set
 - Example: `border: 2px solid Tomato;`
- `rgb(red, green, blue)`
 - Example: `rgb(0, 133, 255)`
- Hex
 - Example: `#ff0000`
- `hsl(hue, saturation, lightness)`
 - Example: `hsl(240, 100%, 50%)`

Borders

- Specifies the style, width and color of an elements border
- `border-style`
 - dotted, dashed, solid, double, groove, ridge, inset, outset, none, hidden
- `border-color`
 - name, Hex, RGB
- `border-width`
 - Can be set in px, pt, cm, em
 - thin, medium, thick
- `border (border-width, border-style, border-color)`
- `border-radius`
 - Used to set rounded corners
- `border-[left,right,bottom,top]-[color,style,width]`
 - All the border properties can be set individually for a specific side
 - Example:

```
border-bottom-style: solid;
border-left-width: thin;
```


Margins

- Creates space around elements outside of any defined borders
- Margins can be specified in px, pt, cm
- `margin (top or all, [right], [bottom], [left])`
 - 4 values specifies all 4 margins
 - 3 values specifies top, right and left, bottom
 - 2 values specifies top and bottom, right and left
 - 1 value makes all margins the same
- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`
- `auto`
 - horizontally centers the element within its container
- `inherit`
 - the margin is inherited from the margin of the parent element

Margin Collapse

- Top and bottom margins sometimes will collapse to the largest specified margin
- Example:

```
h1 {  
    margin-bottom: 50px;  
}  
  
h2 {  
    margin-top: 20px;  
}  
<h1>Hello</h1>  
<h2>World!</h2>
```
- One might expect that the margin between Hello and World would be 70px (50px + 20px)
 - In many cases, the margin will actually only be 50px
 - The assumption is that you wanted a margin of 20px above the h2 tag, and you actually have a margin of 50px, which is greater than what you wanted, so the browser will leave it alone
- Margin Collapse will not happen on left and right margins

CSS Classes

- Allow a designer to define a style once and apply it to any element and as many elements as necessary
- Classes apply to all elements nested within the parent container
- Example:

```
.BodyText {  
    background-color: LightSalmon;  
    font-family: Arial;  
}  
<div class="BodyText">  
    This text should be Arial with a Light  
    Salmon background.  
</div>
```

CSS Subclasses

- Subclasses
 - Set the style for a class within a class
 - Referring to a top level style in a nested element has unpredictable behavior
 - Subclasses give you predictable behavior

- Example:

```
.BodyText {  
    background-color: LightSalmon;  
    font-family: Arial;  
}  
.BodyText .small {  
    font-size: 0.5em;  
}  
<div class="BodyText">  
    This text should be Arial with a Light Salmon  
    background.  
    <div class="small">This text should be 50% of normal  
    size.  
</div>
```

Using CSS in Components

- There are three ways to leverage CSS in Lightning Web Components
 1. Use an inline style
 2. Can be embedded into the Lightning Component
 - a. [componentname].css file
 3. External Style Sheets
 1. Stored as a static resource
 2. Reference to the style sheet in the [componentname].js file

Exercise: Hello World App Style

Goal:

Add some style to your Hello World App.

Action Steps:

1. In the Playground navigate to the **app.css** file.
2. Update the code in the app.css as follows:

```
1  h4 {  
2    font-size: xx-large;  
3    font-family: Arial, Helvetica, sans-serif;  
4    font-weight: bolder;  
5  }  
6  div {  
7    font-family: Arial, Helvetica, sans-serif;  
8    font-size: medium;  
9  }
```

3. The Preview will update automatically as you apply the styles.

Exercise: Hello World LWC Style

Goal:

Add some style to your Hello World Lightning Web Component.

Action Steps:

1. In the Playground navigate to the **helloWorld.css** file.
2. Update the code in the helloWorld.css as follows:

```
1  div {  
2      font-family: 'Times New Roman', Times, serif;  
3      background-color: lightblue;  
4      border-color: yellow;  
5      border-width: 5px;  
6      border-style: solid;  
7  }
```

3. The Preview will update automatically as you apply the styles.

Default Styles

- Lightning Web Components do not have default styles
 - They inherit their style from their parent
- `<template>` cannot have a style applied to it
- To apply a default style to a component, wrap everything in the component in a `<div>` tag and apply a style to the `<div>` tag
- The use of CSS classes is strongly encouraged
 - Element selectors are permitted, but not recommended

Exercise: CSS Class

Goal:

Add a class to the app.css style sheet to ensure all text is formatted the same way, dark blue with a grey background.

Action Steps:

1. In the Playground navigate to the **app.css** file
2. Add a new class to the app.css called GenWatt using the code as follows:

```
.GenWatt {  
    color: ■darkblue;  
    background-color: □lightgrey;  
}
```

3. Navigate to the app.html file and update the code as follows:

```
1  <template>  
2  <div class="GenWatt">  
3      <h4>Hello World App</h4>  
4      <c-hello-world></c-hello-world>  
5      This text should be dark blue with a gray background.  
6  </div>  
7  </template>
```

Exercise: CSS Subclass

Goal:

Add a subclass to the app.css style sheet to allow for formatting of smaller text.

Action Steps:

1. In the Playground navigate to the **app.css** file.
2. Add a new subclass to the GenWatt class named "small" using the code as follows:

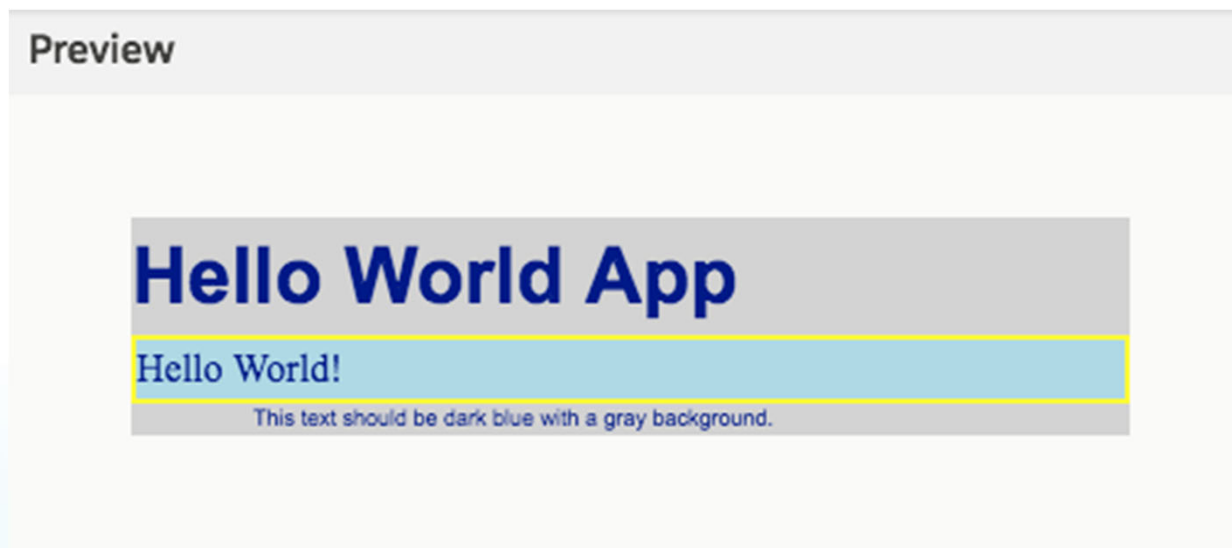
```
.GenWatt .small {  
    font-size: 0.5em;  
}
```

3. Navigate to the **app.html** file and update the code as follows:

```
1  <template>  
2      <div class="GenWatt">  
3          <h4>Hello World App</h4>  
4          <c-hello-world></c-hello-world>  
5          <div class="small">  
6              This text should be dark blue with a gray background.  
7          </div>  
8      </div>  
9  </template>
```

Challenge: Add a Margin

- Add a 50px margin around the outside of all elements within the app
- There should be no margin between the elements within the app
- The app should look like this:



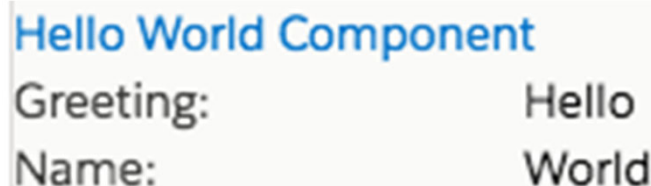
Lightning Design System



- Enables you to build components with the patterns and established best practices that are native to Salesforce
- Includes:
 - Components
 - Allow you to build LWC that look like the built-in Salesforce Components
 - Utilities
 - CSS classes that allow you to apply a simple pattern to components
 - Design Tokens
 - Used in place of hard-coded values so that your components can automatically change if Salesforce changes
 - Icons
 - Created by Salesforce to have a similar look and feel to the built-in Icons
- Automatically available for Lightning Web Components running in Lightning Experience
 - Must be specifically included when using Lightning Out, LC4VF, and Lightning Applications

Challenge: Make a Hello World Tile

- Create a new LWC named helloWorldTile
 - Create an .html, .css and .js file
 - Add the standard JavaScript to the .js file
- Add the helloWorldTile to the app
- Find the CSS for the Base Tile in the Lightning Design System
- Update the .html file so the output looks like this:



Hello World Component

Greeting:	Hello
Name:	World

- The finished app should look like the screenshot on the next page

Challenge: Make a Hello World Tile

Preview

Hello World App

Hello World!

This text should be dark blue with a gray background.

Hello World Component

Greeting: Hello
Name: World

CSS Tips for Components

- The second tag of a component should be a `<div>` tag
- Don't use element selectors, use classes
- Components must be set to 100% width
 - They can be moved to different locations on a Lightning page
- Components must not have a specific width nor a left or right margin.
 - Adding a left or right margin changes the width of a component and can break the layout of the page.
- Don't remove HTML elements from the flow of the document
 - Don't use: `float` or `position`
- Child elements shouldn't be styled to be larger than the root element
 - Bad Example:

```
<div style="height: 50px">  
    <div style="height: 200px"></div>  
</div>
```
- Parent CSS will flow into the component
 - TEST the component thoroughly in all containers

Moving from Playground to Scratch



- Playground does not support all features of Lightning Web Components
- To implement more advanced features we need to move the code to a scratch org using the Salesforce CLI
- The steps to move a Lightning Web Component to a scratch org are:
 1. Save and download the app
 2. Extract the zip file
 3. Create the LWC using

```
sfdx force:lightning:component:create
```
 4. Copy the LWC files into the project directory
 5. Push to Scratch

Exercise: Move helloWorld to Scratch

Goal:

Save the helloWorld Lightning Web Component to your local machine and move it to your Scratch org.

Action Steps:

1. In the Playground, click the Download button.
2. Locate the “Hello World App.zip” file that gets downloaded and extract it.
3. In VS Code, right click on the **lwc** folder and choose Open in Terminal.
4. Run `sfdx force:lightning:component:create -n helloWorld --type lwc` to create the file structure for the component.
5. Open the **lwc\helloWorld** directory on your file system.
6. Copy the following files from the extracted zip file into your **helloWorld** directory:
 - **helloWorld.html**
 - **helloWorld.js**
 - **helloWorld.css**
7. Open a terminal window and run the `sfdx force:source:push` command.

Exercise: Move helloWorldTile to Scratch



Goal:

Save the helloWorldTile Lightning Web Component to your local machine and move it to your Scratch org.

Action Steps:

1. In VS Code, right click on the **lwc** folder and choose Open in Terminal
2. Run `sfdx force:lightning:component:create -n helloWorldTile --type lwc` to create the file structure for the component.
3. Open the **lwc\helloWorldTile** directory on your file system.
4. Copy the following files from the extracted zip file into your helloWorldTile directory:
 - **helloWorldTile.html**
 - **helloWorldTile.js**
 - **helloWorldTile.css**
5. Open a terminal window and run the `sfdx force:source:push` command.

Exercise: Verify helloWorld LWCs



Goal:

Verify that the helloWorld and helloWorldTile Lightning Web Components were successfully pushed to your Scratch org.

Action Steps:

1. In VS Code, right click on the spdex602 folder and choose Open in Terminal.
2. Run `sfdx force:org:open` to login to your Scratch org.
3. Navigate to Setup and search for “Lightning Components”.
4. Verify that the **helloWorld** and the **helloWorldTile** Lightning Components appear in the list.

Lightning Component App



- An independent application with its own, unique URL
- Acts as a container for Lightning Components
 - Can contain both Aura and Lightning Web Components
- Limitations
 - Cannot contain other Lightning Apps
 - Cannot be added to the Lightning App Launcher
- Benefits
 - Can be launched by a URL path
 - Can be easily styled independently
 - Can accept parameters passed in through the URL

Using LWC in Aura Components



- It is possible to utilize Lightning Web Components in a Lightning Aura Component
 - Lightning Web Components cannot contain Aura Components
- It's necessary in many cases
 - Refer to the slide "No Support for Lightning Web Components"
- The Aura syntax for referencing the components is used:
 - Example: helloWorld component
 - LWC syntax `<c-hello-world>`
 - Aura syntax: `<c:helloWorld>`
 - A Lightning Component App must reference Lightning Web Components using the Aura syntax

Exercise: Hello World App



Goal:

Create your first Lightning App using VS Code.

Action Steps:

1. In VS Code, click View -> Command Palette and type **SFDX: Create Lightning App**
2. Name the app **helloWorldApp**
3. Select the default location of **force-app\main\default\aura**
4. Update the code in the helloWorldApp.app as follows:

```
1 <aura:application >  
2     <c:helloWorld/>  
3 </aura:application>
```

5. Save helloWorldApp and push it out to your scratch org.

Exercise: Add helloWorldTile to the App

Goal:

Add the helloWorldTile component to the helloWorldApp.

Action Steps:

1. In VS Code, update the code in the **helloWorldApp.app** as follows:

```
1 <aura:application >
2     <div>
3         <c:helloWorldTile></c:helloWorldTile>
4     </div>
5     <div>
6         <c:helloWorld></c:helloWorld>
7     </div>
8 </aura:application>
```

2. Save the helloWorldApp and push it out to your scratch org.

Lightning Paths



- <https://<yourDomain>.lightning.force.com/<yourNamespace>/<yourAppName>.app>
 - The namespace for your apps will default to “c”
 - Signifies a custom application
- Your helloWorldApp will be at this path
<https://<yourDomain>.lightning.force.com/c/helloWorldApp.app>
- App Names are case insensitive

Component Configuration File

- Every LWC must have a configuration file
 - Cannot be pushed or deployed without one
- XML file that defines the metadata values for the component that impact
 - The API version
 - Where the component can be used
 - How the component displays in Setup
- Will follow this naming convention [component name].js-meta.xml

-meta.xml Options

- `<apiVersion>48.0</apiVersion>`
- `<isExposed>true</isExposed>`
 - false by default
 - When set to true, the component can be seen in Lightning App Builder
- `<masterLabel>Hello World</masterLabel>`
 - The label that displays in Setup
- `<description></description>`
 - The text that appears in Setup in the list of Lightning Components and also on hover in the Lightning App Builder

<targets>

- <targets>
 - Specify what types of pages the component can be used on
 - A component can have multiple targets
 - lightning__AppPage
 - lightning__HomePage
 - lightning__RecordPage
 - lightningCommunity__Page
 - This code would allow a component to be displayed on the Home Page, in a Lightning App or on a Record page

```
<targets>  
  <target>lightning__AppPage</target>  
  <target>lightning__HomePage</target>  
  <target>lightning__RecordPage</target>  
</targets>
```




Exercise: Update helloWorld.js-meta.xml

Goal:

Update the helloWorld.js-meta.xml file to have a title, description and be available for use in Lightning Experience pages.

Action Steps:

1. In VS code, update the code in the **helloWorld.js-meta.xml** to be:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata"
3      fqn="helloWorld">
4      <apiVersion>46.0</apiVersion>
5      <isExposed>true</isExposed>
6      <masterLabel>Hello World</masterLabel>
7      <description>Hello World</description>
8      <targets>
9          <target>lightning__AppPage</target>
10         <target>lightning__HomePage</target>
11         <target>lightning__RecordPage</target>
12     </targets>
13 </LightningComponentBundle>
```

2. From the terminal in VS Code, run `sfdx force:source:push` to push the changes to Scratch.

Lightning Pages & Tabs



- Good for building custom functionality that isn't directly tied to a specific Salesforce object
- Cannot display
 - Lightning Record Pages
 - Lightning Home Pages
- Can be used to display a Lightning Web Component that is embedded in a Lightning Page
- Lightning Page Tabs
 - Used to display Lightning Pages

Exercise: Add Hello World to a Lightning Page



Goal:

Add the Hello World LWC to a Lightning Page and display it in a Lightning Tab.

Action Steps:

1. In your scratch org, navigate to Setup and search for “Lightning App Builder”.
2. Click the New button to create a new Lightning Page.
3. Choose to create an App Page and click the Next button.
4. Set the Label to “Hello World” and click the Next button.
5. Choose “Two Regions” as the layout and click the Finish button.
6. Drag the Hello World component to the left region of the page.
7. Save the page and Activate the page.
 - a. Set the App Name to: **Hello World**
 - b. Activate the page for all users.
 - c. Choose an Icon of your liking.
 - d. Click the Lightning Experience tab.
 - a. Add the page to the GenWatt Sales app.
 - b. Move the Hello World tab to the top.
 - e. Click the Save button.
8. Using the App Launcher, navigate to the GenWatt Sales app and verify that the Hello World tab is the first tab.

Exercise: Add Hello World to Account Page



Goal:

Add the Hello World LWC to the existing GenWatt Account Lightning Record Page in your scratch org.

Action Steps:

1. In your scratch org, navigate to Setup -> Object Manager -> Account -> Lightning Record Pages.
2. Open the GenWatt Account page for editing.
3. Click on the Tabs component in the left region of the page.
4. You will see Tabs appear in the property pane on the far right.
 - a. Click on the Add Tab button to add a new Tab.
 - b. Change the new Tab Label to **Custom** and enter **Hello World** as the Custom Tab Label.
 - c. Set the Default Tab to Hello World.
 - d. Drag the Hello World component onto the Hello World tab.
5. Save the page.
6. Verify that you see the Hello World component when viewing an Account record.
7. From the terminal in VS Code, run `sfdx force:source:pull` to pull the changes from Scratch.

JavaScript

- JavaScript is the programming language of HTML and the Web
- Used to deliver client-side automation in LWC.
- Third-party JavaScript libraries can be used in LWC by uploading them as a static resource.

JavaScript in Components



- Every component must have a JavaScript file named [component name].js
 - Contains client-side methods to
 - handle events in the component
 - communicate with the server
 - declare variables
 - interact with the DOM
 - Must have an `import{}` statement
 - Must have an `export{}` statement
- Can import external JavaScript ES6 modules

import{ } and export{ }

- `import{ }`
 - Used to reference code in an external JavaScript file
 - Every Lightning Web Component must import `LightningElement`
 - `import { LightningElement } from 'lwc';`
 - `LightningElement` is a wrapper of the standard HTML element
- `export{ }`
 - Used to allow other modules to reference the code in the class
 - Every Lightning Web Component must export a default class that extends `LightningElement`
 - This allows the component to be embedded in other components
 - `export default class MyComponent extends LightningElement {}`

Document Object Model (DOM)



- When a web page is loaded, the browser creates an object model for every element on the page
 - The model is structured like a tree with branches
 - <HTML> - <Body> - <div> - <p>
- JavaScript can consume and modify the DOM
 - This is called Dynamic HTML
- JavaScript can:
 - Change all the HTML elements in the page
 - Change all the HTML attributes in the page
 - Change all the CSS styles in the page
 - Remove existing HTML elements and attributes
 - Add new HTML elements and attributes
 - React to all existing HTML events in the page
 - Can create new HTML events in the page

Chrome DevTools

- Built into the Chrome Browser
 - Element Viewer / DOM Viewer
 - CSS Editor
 - Performance Analyzer
 - Memory Analyzer
- JavaScript Console
 - Similar to Execute Anonymous in the Salesforce Developer Console
 - Can execute and run JavaScript
 - Can log information from JavaScript within an executing web page

JavaScript Common Uses



- Declaring and storing variables
- Looping
- Importing external files
- Responding to Events
- Interacting with Salesforce server
- Changing HTML
- Changing Styles

JavaScript Loops

- **For Loop**

```
for (i=0; i<5; i++) {  
    console.log('The number is: ' + i);  
}
```

- **While Loop**

```
var i = 0;  
while (i < 5) {  
    console.log('The number is: ' + i);  
    i++;  
}
```

- **Do While Loop**

```
var i = 0;  
do {  
    console.log('The number is: ' + i);  
    i++;  
}  
while (i < 10);
```

- Object Loop

```
var person = {fname:"John",  
lname:"Doe", age:25};  
var x;  
for (x in person) {  
    console.log (person[x]);  
}
```

- `break;`
 - Exits the loop completely
- `continue;`
 - Skips to the next iteration of the loop

Exercise: Chrome Console Loops

Goal:

Execute loops within the Chrome Console.

Action Steps:

1. Open a Chrome browser window.
2. Click ⋮ -> More tools -> Developer tools. Click on the Console tab.
3. Write a for loop and log each iteration to the Console using `console.log`.
4. Write a while loop and log each iteration to the Console using `console.log`.
5. Write a do while loop and log each iteration to the Console using `console.log`.

Static Resources



- Static resources allow you to upload content that you can reference in a Visualforce page or Lightning Component
- Can include archives (such as .zip and .jar files), images, style sheets, JavaScript, and other files.
 - You can package a collection of related files into a directory hierarchy and upload that hierarchy as a .zip or .jar archive.
- Preferable to uploading a file to the Documents tab
 - You can reference a static resource by name in page markup by using the \$Resource global variable instead of hard coding document IDs
- Preferable to an external webserver for security and performance reasons
- Cannot be used in the Playground because there is no authentication in the playground

External Style Sheets

- Allow one style sheet to be created and used across an unlimited number of web pages
 - Lightning Web Components can reference external style sheets
- Reduces the need to duplicate styles within multiple components
- JavaScript is used to reference an external style sheet in Lightning Web Components
 - The style sheet should be loaded as a static resource
- Best Practices for External CSS
 - To avoid conflicts with other CSS, each external CSS file should have its own unique namespace (class name)
 - When using external CSS classes, wrap the elements in a <div> tag with the external class namespace

Exercise: Hello World LWC Style



Goal:

To avoid conflicts with your external style sheet, update the css for the Hello World Lightning Web Component

Action Steps:

1. Using VS Code, navigate to the helloWorld.css file
2. Update the code in the helloWorld.css as follows:

```
.defaultStyle {  
    font-family: 'Times New Roman', Times, serif;  
    background-color: lightblue;  
    border-color: yellow;  
    border-width: 5px;  
    border-style: solid;  
}
```

3. From the terminal in VS Code, run `sfdx force:source:push` to push the changes to Scratch

Exercise: Static Resource



Goal:

Create an external style sheet and add it as a static resource.

Action Steps:

1. Using a text editor, such as notepad, create a new file called **GenWatt2.css**
2. Add the following code to the file:

```
.GenWatt2 {  
    border-width: thick;  
    border-color: red;  
    border-style: solid;  
}
```
3. Save the file as **GenWatt2.css**
4. In your scratch org, navigate to Setup -> Custom Code -> Static Resources.
5. Create a new Static Resource named **GenWatt2**.
 - a. Choose the **GenWatt2.css** file as the file.
 - b. Cache Control: **Public**
6. Save the static resource.
7. From the terminal in VS Code, run `sfdx force:source:pull` to pull the changes to Scratch.

Component Lifecycle



- Lightning Web Components have a lifecycle managed by the framework
- `constructor()`
 - Fires when the component is created
- `connectedCallback()`
 - Fires when a component is inserted into the DOM
 - `disconnectedCallback()` fires when a component is removed from the DOM
- `renderedCallback()`
 - Fires when a component has finished rendering
- `errorCallback()`
 - Captures errors in all children of a component

constructor()

- Fires when the component is created
- Must follow certain rules
 - The first line of the constructor must be
 - `super();`
 - Don't use `document.write()`
 - Don't use `document.open()`
 - Don't inspect attributes or children because they don't exist
 - Don't inspect public properties because those aren't set yet
 - Don't add attributes

Exercise: Loading a Static Resource

Goal:

Load the GenWatt2 static resource into the helloWorld component.

Action Steps:

1. In VS Code, open the **helloWorld.js** file.
2. Update the code so that it looks like the code below:

```
1  import { LightningElement } from 'lwc';
2  import GenWatt2 from '@salesforce/resourceUrl/GenWatt2';
3  import { loadStyle } from 'lightning/platformResourceLoader';
4
5  export default class HelloWorld extends LightningElement {
6      constructor() {
7          super();
8          loadStyle(this, GenWatt2)
9              .then(() => { /* callback */});
10     }
11 }
```

Exercise: External Style Sheet



Goal:

Use the GenWatt2 stylesheet in the helloWorld component.

Action Steps:

1. In VS Code, open the **helloWorld.html** file.
2. Update the code so that it looks like the code below:

```
1  <template>
2  |    <div class="GenWatt2">Hello World!</div>
3  </template>
```

3. From the terminal in VS Code, run `sfdx force:source:push` to push the changes to Scratch.
4. Preview the Hello World App and verify it looks like the screenshot below:

Hello World Component

Greeting:

Hello

Name:

World

Hello World!

Public Properties



- Are declared in the [component name].js file
- A parent of the component can access the public properties
 - Camel to Kebab case `sampleProperty` becomes `sample-property`
- Are Reactive
 - If the value of the property changes, the component rerenders
 - When a component rerenders, all expressions used in the template are reevaluated
- To utilize public properties
 - Must import api from lwc
 - Declare the property using `@api`
- Example:

```
1  import { LightningElement, api } from 'lwc';
2  export default class MyComponent extends LightningElement {
3    |    @api sampleProperty = 'Default Value';
4  }
```


Exercise: Public Property

Goal:

Add a name property to the helloWorld component and set it in the Hello World App.

Action Steps:

1. In VS Code, open the **helloWorld.js** file.
2. Update the code so that it looks like the code below:

```
1  import { LightningElement, api } from 'lwc';
2  import GenWatt2 from '@salesforce/resourceUrl/GenWatt2';
3  import { loadStyle } from 'lightning/platformResourceLoader';
4
5  export default class HelloWorld extends LightningElement {
6      @api name = 'World';
7      constructor() {
8          super();
9          loadStyle(this, GenWatt2)
10             .then(() => {});
11     }
12 }
```

Exercise: Public Property (continued)



Action Steps:

3. In VS Code, open the **helloWorld.html** file.
4. Update the code so that it looks like the code below:

```
1 <template>
2 |   <div class="GenWatt2">Hello {name}!</div>
3 </template>
```

5. Update the code in the **helloWorldApp** so that it looks like the code below:

```
1 <aura:application >
2   <div>
3     <c:helloWorldTile></c:helloWorldTile>
4   </div>
5   <div>
6     <c:helloWorld name="Galaxy"></c:helloWorld>
7   </div>
8 </aura:application>
```

6. From the terminal in VS Code, run `sfdx force:source:push` to push the changes to your scratch org.
7. Run `sfdx force:org:open` to open scratch org.
8. Preview the Hello World App and verify it says Hello Galaxy!

Properties in Lightning App Builder



- Public properties declared on a component are not automatically available in the Lightning App Builder
- Can be exposed using `<targetConfig>` in the Configuration file
- Each `<target>` can (should) have a corresponding `<targetConfig>`

Exercise: Property for App Builder



Goal:

Expose the name property of the helloWorld component to the Lightning App Builder.

Action Steps:

1. In VS Code, open the **helloWorld.js-meta.xml** file
2. Add the code below after the **</targets>** tag and before the **</LightningComponentBundle>** tag:

```
<targetConfigs>
  <targetConfig targets="lightning__RecordPage, lightning__AppPage, lightning__HomePage">
    <property name="name" type="String" default="World"
      label="Name" description="Sets the name of the party to greet"/>
  </targetConfig>
</targetConfigs>
```

3. Save the file and run `sfdx force:source:push` to push the changes to Scratch.

Exercise: Test Name Property



Goal:

Test that you can set the name property of the helloWorld component in the Lightning App Builder.

Action Steps:

1. In VS Code, run `sfdx force:org:open` to open scratch org.
2. Navigate to Setup and search for “Lightning App Builder”.
3. Edit the GenWatt Account page.
4. In the App Builder, click on the Hello World tab and then click on the Hello World component.
5. In the properties pane on the right, set the value for the Name field to “Universe”.
6. Save the page.
7. Navigate to any Account record and verify that the Hello World component displays “Hello Universe”.

Lightning Data Service



- A method of retrieving data from the database without having to write custom Apex code
- Works natively with these Lightning Components
 - lightning-record-form
 - lightning-record-edit-form
 - lightning-record-view-form
- Built on top of the User Interface API
- UI API
 - Returns both data and metadata in a single response
 - Respects all Security Settings

lightning-record-form



- The easiest LDS form to use
- Inherits all the form settings from the page layouts
- Little customization possible
- Good for:
 - Creating and editing records using the standard behavior
 - Viewing existing records

Walkthrough: lightning-record-form



- Create a new component called oppRecordForm to display a lightning-record-form for an Opportunity record
- Set the properties for the form
- Set the configuration values
- Add the component to a tab on the Opportunity page layout

Nesting Components

- Components can act as containers for other components
- Nesting occurs when one component is placed inside another component
 - The nested component is the component placed within another component
 - The nested component is referred to as a child component
 - The component that has a nested component within it is referred to as the parent component
- There is no published limit of number of nested levels
 - Performance problems have been reported in nested level 3 and deeper

Events

- Typically triggered by a user interacting with the user interface
 - Can also be triggered by
 - The Streaming API
 - Platform Events
- All Lightning Web Components implement the EventTarget interface which allows them to
 - Listen for events
 - Handle events
 - Create and Dispatch events
- Used to pass information from child to parent
 - Parents should pass information to children using properties on the child
- Events raised by a component
 - By default are available to their parent only
 - Can be configured to bubble all the way up the DOM
- Lightning Web Components support Publish-Subscribe
 - A component can raise an event and many other components can subscribe to that event, regardless of where they are in the DOM tree

Walkthrough: formToggle component

- Create a new component called formToggle to allow toggling between edit mode and view mode
- Set the configuration values
- Add the component to a tab on the Opportunity page layout
- Identify and correct any bugs

- Create a new component called oppRecordViewForm to display a lightning-record-view-form for an Opportunity record
- Set the properties for the form
- Set the configuration values
- Add the component to a tab on the Opportunity page layout

- Create a new component called oppRecordEditForm to display a lightning-record-edit-form for an Opportunity record
- Set the properties for the form
- Set the configuration values
- Add the component to a tab on the Opportunity page layout

Business Scenario: Opportunities

GenWatt hoped that upgrading to the Lightning Experience would allow them to add filtered related lists to their page layouts, but the Lightning Experience does not provide filtered related lists.

GenWatt has asked that you develop a filtered related lists for Opportunities on the Account record. GenWatt would like to be able to filter the list by All, Open Opportunities, Closed Opportunities, Closed – Won Opportunities and Closed – Lost Opportunities. They would also like to understand the number of Opportunities with each status and the total amount of the Opportunities.

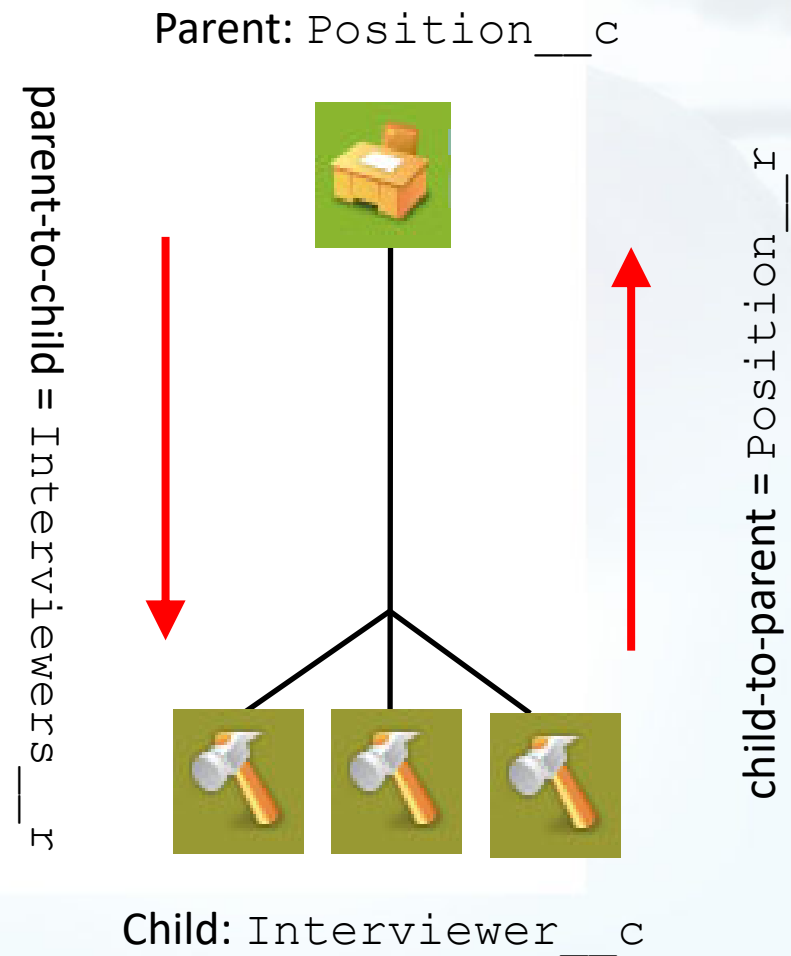
Technical Scenario: Opportunities

1. Develop a Lightning Component to display an Opportunity
 - a. Implement functionality to allow the user to navigate to the full Opportunity record
2. Develop an Apex class to retrieve the Opportunity records related to an Account
3. Develop a Lightning Component to display a related list of Opportunities
 - a. Implement the routines to update the list based on the filter selected
4. Add the Opportunity List Component to the Lightning Pages for Accounts

- SOQL (Salesforce Object Query Language)
 - Very similar to SQL
 - Query only
 - Cannot be used to modify or create data
 - Cannot be used to modify object structures
 - No ad hoc table joins allowed
 - All joins based on pre-defined object relationships
 - Case Insensitive
- SOSL (Salesforce Object Search Language)
 - Duplicates the functionality of the global search
 - Can search for the search phrase across multiple objects at once
 - Case Insensitive

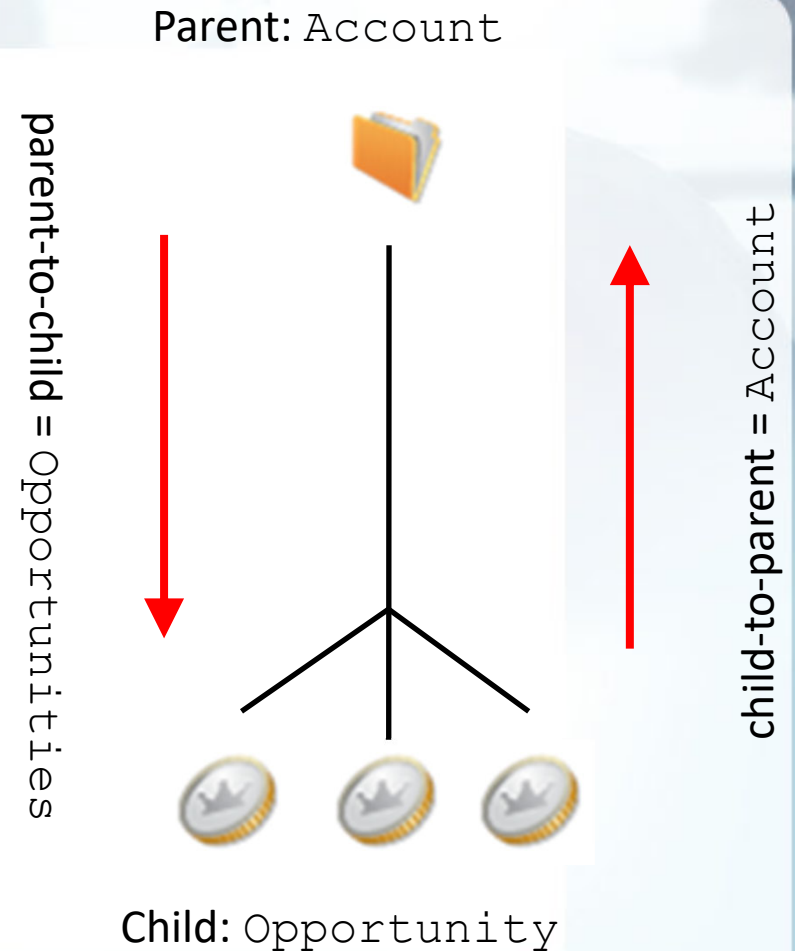
sObject Relationships

- Standard relationship
 - Created by Salesforce.com and cannot be modified
- Custom relationship
 - Created by a developer
 - Can be modified by a developer
 - Both directions are appended with `__r`
- A relationship has different names depending on the direction you are traversing:
 - Parent-to-child: plural version of the child object.
 - Child-to-parent: singular version of the parent object.



sObject Relationships in Code

- Relationships in Apex and SOQL are written using dot notation
- Allows a developer to query multiple tables without an explicit table join
- Can traverse up to five levels parents
- Can include only one level of children



```
Opportunity.Account.Industry = 'Mining'
```

SOQL Queries: Right Outer Join



Position__c

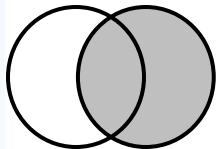
Id	Name	Dept__c
a05S0000000WZeY	Programmer	Engineering
a05300000050Krc	Account Executive	Sales
a05S0000000X2eY	Billing Assistant	Finance
a05300000050KrT	Product Manager	Engineering

Job_Application__c

Name	Position__c	Status__c
APP-0069	a05S0000000WZeY	Open
APP-0070		Open
APP-0071	a05S0000000WZeY	Closed
APP-0072	a05300000050Krc	Open

Get job applications with associated position data

```
SELECT Name, Position__r.Dept__c FROM Job_Application__c
```



List<Job_Application__c>

Name	Position__r.Dept__c
APP-0069	Engineering
APP-0070	
APP-0071	Engineering
APP-0072	Sales

SOQL Queries: Left Outer Join

Position__c

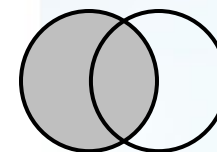
Id	Name	Dept__c
a05S0000000WZeY	Programmer	Engineering
a05300000050Krc	Account Executive	Sales
a05S0000000X2eY	Billing Assistant	Finance
a05300000050KrT	Product Manager	Engineering

Job_Application__c

Name	Position__c	Status__c
APP-0069	a05S0000000WZeY	Open
APP-0070		Open
APP-0071	a05S0000000WZeY	Closed
APP-0072	a05300000050Krc	Open

Get positions and related job application data

```
SELECT Name, (SELECT Name FROM Job_Applications__r)
FROM Position__c
```



List<Position__c>

Name	Job_Applications__r
Programmer	(Related List)
Billing Assistant	
Account Executive	(Related List)
Product Manager	

List<Job_Application__c>

Name
APP-0069
APP-0071

Name
APP-0072

SOQL Queries: Semi-Join (Left Inner Join)



Position__c

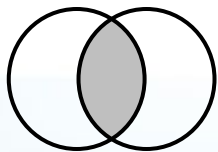
Id	Name	Dept__c
a05S0000000WZeY	Programmer	Engineering
a05300000050Krc	Account Executive	Sales
a05S0000000X2eY	Billing Assistant	Finance
a05300000050KrT	Product Manager	Engineering

Job_Application__c

Name	Position__c	Status__c
APP-0069	a05S0000000WZeY	Open
APP-0070		Open
APP-0071	a05S0000000WZeY	Closed
APP-0072	a05300000050Krc	Open

Only get positions with related job applications

```
SELECT Name FROM Position__c
WHERE Id IN (SELECT Position__c FROM Job_Application__c)
```



List<Position__c>

Name
Programmer
Account Executive

SOQL Queries: Right Inner Join



Position__c

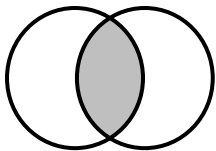
Id	Name	Dept__c
a05S0000000WZeY	Programmer	Engineering
a05300000050Krc	Account Executive	Sales
a05S0000000X2eY	Billing Assistant	Finance
a05300000050KrcT	Product Manager	Engineering

Job_Application__c

Name	Position__c	Status__c
APP-0069	a05S0000000WZeY	Open
APP-0070		Open
APP-0071	a05S0000000WZeY	Closed
APP-0072	a05300000050Krc	Open

Only get job applications with related positions

```
SELECT Name, Status__c, Position__r.Name  
FROM Job_Application__c WHERE Position__c != null
```



List<Job_Application__c>

Name	Status__c	Position__r.Name
APP-0069	Open	Programmer
APP-0071	Closed	Programmer
APP-0072	Open	Account Executive

SOQL Queries: Right Anti-Join



Position__c

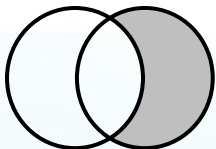
Id	Name	Dept__c
a05S0000000WZeY	Programmer	Engineering
a05300000050Krc	Account Executive	Sales
a05S0000000X2eY	Billing Assistant	Finance
a05300000050KrcT	Product Manager	Engineering

Job_Application__c

Name	Position__c	Status__c
APP-0069	a05S0000000WZeY	Open
APP-0070		Open
APP-0071	a05S0000000WZeY	Closed
APP-0072	a05300000050Krc	Open

Only get job applications without related positions

```
SELECT Name, Status__c FROM Job_Application__c
WHERE Position__c = null
```



List<Job_Application__c>

Name	Status__c
APP-0070	Open

SOQL Queries: Left Anti-Join

Position__c

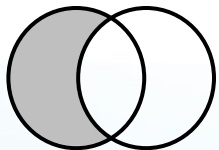
Id	Name	Dept__c
a05S0000000WZeY	Programmer	Engineering
a05300000050Krc	Account Executive	Sales
a05S0000000X2eY	Billing Assistant	Finance
a05300000050KrT	Product Manager	Engineering

Job_Application__c

Name	Position__c	Status__c
APP-0069	a05S0000000WZeY	Open
APP-0070		Open
APP-0071	a05S0000000WZeY	Closed
APP-0072	a05300000050Krc	Open

Only get positions without related job applications

```
SELECT Name FROM Position__c WHERE Id
NOT IN (SELECT Position__c FROM Job_Application__c)
```



List<Position__c>

Name
Billing Assistant
Product Manager

Multi-Level Relationships



Relationship names can be chained together to access parent or child records

- Up to five levels deep for child-to-parent relationships.
- One level deep parent-to-child relationship.

Parent Record Example:

```
SELECT o.Opportunity.Account.Name  
FROM OpportunityLineItem o
```

Children Records Example:

```
SELECT o.Name,  
       (Select PricebookEntryId FROM  
        OpportunityLineItems) FROM Opportunity o
```

Important SOQL Keywords



- ORDER BY
 - ASC or DESC
- LIMIT
 - Limits the records returned
 - When used with ORDER BY, sorts first then applies limit
- GROUP BY
 - ROLLUP to calculate subtotals
- IN
- LIKE
 - % is wildcard
- FOR UPDATE
 - Locks the returned record from being updated by another request
- ALL ROWS
 - Returns both active records and records in the Recycle Bin

Exercise: Force.com SOQL Reference Guide



Goal:

Create a bookmark to the Force.com SOQL and SOSL Reference.

Action Steps:

1. In a new browser window, navigate to <http://developer.salesforce.com>
2. In the Search bar, type in SOQL Reference.
3. The first search result should be the guide.
4. Click on the link and create a bookmark to the page that comes up.

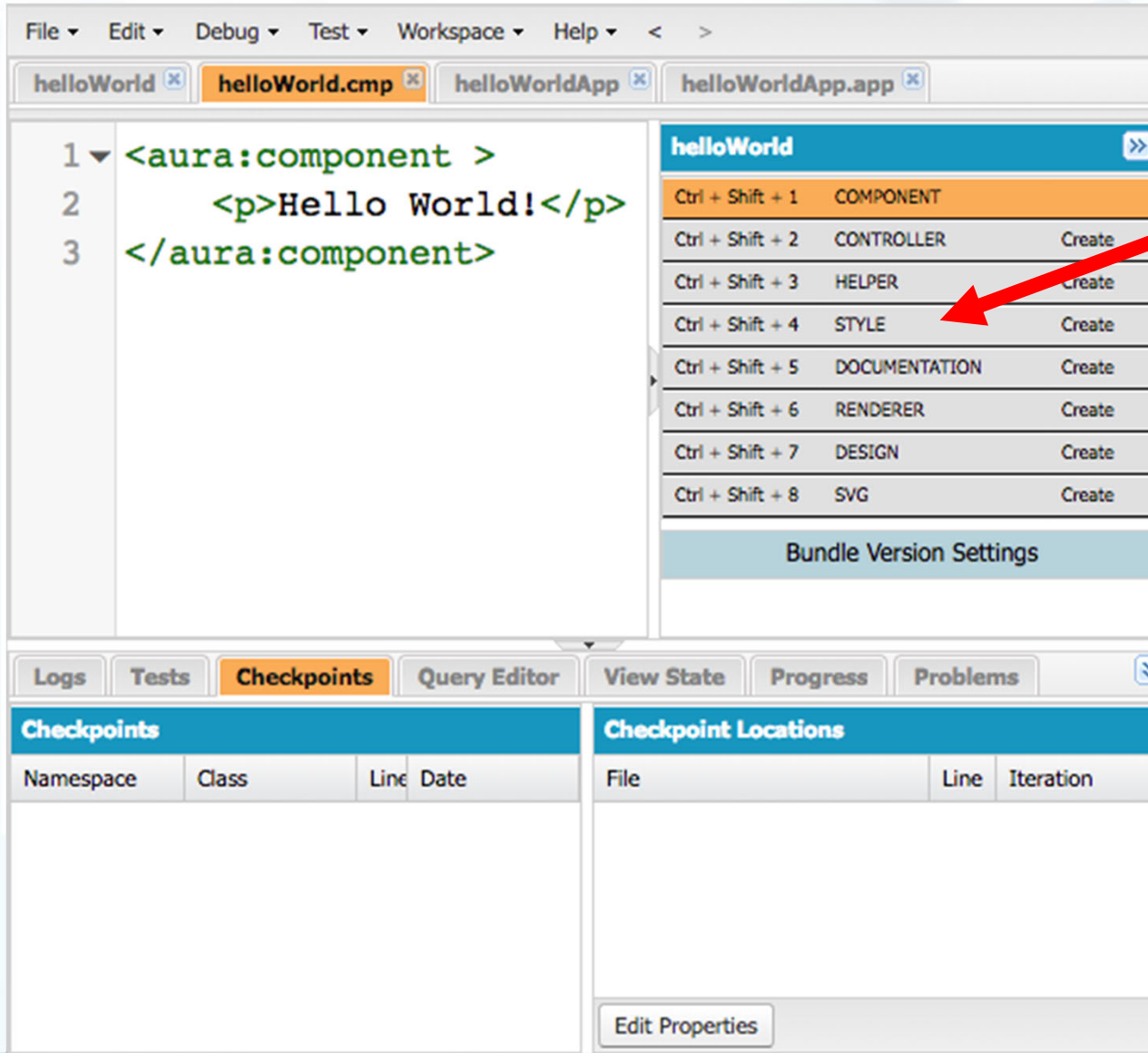
Developer Console



- Native, browser-based development tool
- Create and edit source code for:
 - Apex Triggers and Classes
 - Visualforce Pages & Components
 - Lightning Pages & Lightning Aura Components
 - CSS, Javascript, SVG, XML
 - Cannot be used to develop Lightning Web Components
- Execute SOQL & SOSL
- Debug
- Test
- Execute Anonymous
- Supports some command line functions

Developer Console

Resources



The screenshot shows the Developer Console interface. The top menu bar includes File, Edit, Debug, Test, Workspace, and Help. The tab bar shows four tabs: helloWorld, helloWorld.cmp (selected), helloWorldApp, and helloWorldApp.app. The main editor area displays the following XML code:

```
1 <aura:component >
2   <p>Hello World!</p>
3 </aura:component>
```

On the right side, the 'Resources' panel for 'helloWorld' is open. It lists various resource types with their corresponding keyboard shortcuts and a 'Create' button:

Shortcut	Resource Type	Action
Ctrl + Shift + 1	COMPONENT	Create
Ctrl + Shift + 2	CONTROLLER	Create
Ctrl + Shift + 3	HELPER	Create
Ctrl + Shift + 4	STYLE	Create
Ctrl + Shift + 5	DOCUMENTATION	Create
Ctrl + Shift + 6	RENDERER	Create
Ctrl + Shift + 7	DESIGN	Create
Ctrl + Shift + 8	SVG	Create

Below the list is a 'Bundle Version Settings' section. At the bottom of the console, there are tabs for Logs, Tests, Checkpoints (selected), Query Editor, View State, Progress, and Problems. The 'Checkpoints' tab is active, showing two panels: 'Checkpoints' and 'Checkpoint Locations'. The 'Checkpoints' panel has columns for Namespace, Class, Line, and Date. The 'Checkpoint Locations' panel has columns for File, Line, and Iteration. An 'Edit Properties' button is located at the bottom right of the console.

Walkthrough: SOQL Queries



Goal:

Use the SOQL Query Builder and the Salesforce Developer Console and to perform the queries below

Action Steps:

1. Navigate to <https://soqlbuilder.herokuapp.com/> to help build the queries
2. Within the Developer Console, navigate to the Query Editor window
3. Select all the Accounts and any related Opportunities
4. Select all Accounts and all related Cases and all related Contacts
5. Select all Cases, the name of the Account associated with the Case and the Name of the Contact associated with the Case
6. Select all Products and any Opportunities they are related to

Apex

- A proprietary development language created by salesforce.com
 - Syntax for language is a combination of Java and C#
 - Only runs on the Lightning Platform
- Not another name for Visualforce, a different technology
- Supports most object-oriented constructs
- Very limited in scope
 - Primary functionality available is math, logic and data manipulation
- Fits into the Controller level of MVC
 - No ability to interact directly with the screen
 - Can be used as a custom controller for a Visualforce page
 - Can be a custom controller for a Lightning Component

- A Java or C# development skill set makes the easiest transition to Apex development
 - Any developer with object-oriented programming experience should be able to pick it up in 1-2 months
- Administrators typically cannot make the leap to writing Apex
- Developer Console is the development environment of choice
- Can be used to create:
 - Triggers
 - Classes
 - Visualforce Controllers
 - Lightning Component Controllers
 - Batch Process
 - Custom Web Services
 - Custom Email Handlers

When to use Apex

- Current Use Cases
 - As a controller for a Lightning Component
 - To delete records
 - To stop the delete of a record
 - To automate manually sharing of records
 - Complex validation rules
 - To call an external web service synchronously
 - To develop custom webs services
 - To perform batch processing
 - To run scheduled jobs
 - As a controller for a Visualforce page
 - When you can't do it declaratively!
- Historic Use Cases that Process Builder can Accomplish
 - To update fields on a related record
 - To create new records
 - To update records with values from another record
 - To update Lookup fields on a record

Development Path



1. Attempt Declarative
 - i. Page Layouts / Record Types
 - ii. Required Fields
 - iii. Validation Rules
 - iv. Workflow
 - v. Visual Workflow
 - vi. Process Builder
2. Attempt Apex only
 - i. Triggers behind the scenes
 - ii. Batch Apex to update records
3. Attempt to use Custom Lightning Components
4. Attempt to use Heroku
5. Use an external programming language and deploy on Salesforce Canvas

How to Deploy Apex



- Apex can be deployed as either a class or a trigger
- Triggers
 - More procedural than object oriented
 - Bound to specific data events on a specific object
 - Code is invoked when a data event fires
- Classes
 - Support most object-oriented constructs
 - Must be invoked by another process

Apex Key Concepts



- Bulk Processing
 - Triggers are written to process records in bulk / batch
 - Instead of firing one time per record, the trigger needs to be coded to fire one time per every two hundred records
 - “Bulkify” – writing a trigger to properly support multiple records
- Limits
 - Because Salesforce is a multi-tenant environment, you are limited in the volume of resources you can consume
- Unit Tests
 - All Apex code must be covered by Unit Tests
 - 75% of all code must be covered
 - Every trigger must have at least 1% test coverage
- With Sharing / Without Sharing
 - Apex code can be configured to either respect or ignore security
 - Default behavior is to ignore security

Limits

- Force.com has two types of limits
 - Cumulative Limits
 - Governor Limits
- Cumulative Limits
 - Salesforce keeps track of limits over a rolling 24 hours
 - Examples:
 - Total # of inbound API Calls per day
 - Total # of mass email messages per day
 - Total # of web to lead Leads per day
- Governor Limits
 - Per transaction and reset after each transaction
 - No documented total limit
 - Examples:
 - A SOQL query can return no more than 50,000 rows
 - A DML statement can process no more than 10,000 rows
 - The total heap size of a transaction is limited to 6 MB

Apex Data Types



- Apex is strongly typed
- All variable types are objects
- Major types
 - Primitives (`Integer`, `Date`, `DateTime`, `Long`, `String`, `Boolean`...)
 - sObjects (`sObject`, or `Account`, `Position__c`, ...)
 - Collections (`List`, `Set`, or `Map`)
 - An object created from user- or system-defined classes
 - Null (for the `null` constant)
 - Object

Primitive Data Types



Apex uses the primitive data types supported by SOAP

- Boolean
- Date
- Datetime
- Time
- ID (18 character, although can accept 15-digit and convert it)
- Integer, Long, Double, Decimal
- String
- Blob

sObject Data Type



- Standard or Custom Objects as defined in the declarative interface
 - Custom objects always end in __c
 - Example: Invoice__c
- Characteristics
 - Hold Data
 - Can be returned from a SOQL query
 - Can be created in memory programmatically
 - Respect security
 - Have Fields
 - Always have an ID Field
 - Cannot be set programmatically
 - If the ID field is not null, it indicates the record already exists in the database
 - Custom fields always end in __c
 - Example: Invoice__c.Amount__c

Apex & Metadata



- Apex is strongly typed to Salesforce metadata.
 - All sObjects and fields can be accessed using Apex.
- **Apex is Metadata aware.**
 - Declarative features referenced by Apex are protected from changes
 - Example: An administrator will not be able to delete a field that is referenced within Apex code
 - Pro: It greatly reduces the likelihood of code being broken
 - Con: It makes it difficult to remove unwanted functionality after it has been deployed
- Upgrades
 - Apex is not auto-upgraded with each new release
 - Apex is saved with a specific version of the API and the functionality of each version of the API is preserved with each release

Apex Collections

- Set
 - An unordered list of unique keys
- List
 - An ordered list of objects
 - Can only sort ascending
 - Can be nested
- Map
 - A key, value pair
 - Essentially a set and a list
 - Can be nested
 - Used primarily for in-memory table joins

Looping

- Apex supports five types of loops:
 - `do{statement}`
`while(Boolean_condition);`
 - `while(Boolean_condition) statement;`
 - `FOR(initialization;`
`Boolean_exit_condition; increment)`
`statement;`
 - `FOR(variable : list_or_set) statement;`
 - `FOR(variable : [inline_soql_query])`
`statement;`
- All loops support following methods:
 - `break`: exits the loop fully
 - `continue`: exits the current iteration and moves to the next iteration of the loop

Do-While & While Loops

- The `do-while` loop repeatedly executes as long a particular Boolean condition remains true.

```
1 Integer count = 1;
2 do{
3     System.debug(count);
4     count++;
5 } while(count < 11);
```

- The `while` loop checks the condition before the first iteration.

```
1 Integer count = 1;
2 while(count < 11) {
3     System.debug(count);
4     count++;
5 }
```

For Loops

- Iterating with an integer

```
1  FOR(Integer i = 0; i < 10; i++) {  
2      System.debug(i+1);  
3  }
```

- Iterating through a List

```
1  List<Integer> myInts = new List<Integer>();  
2  myInts.add(1);  
3  FOR(Integer i : myInts) {  
4      System.debug(i);  
5  }
```

- Iterating through a query result

```
1  FOR(Account a : [Select Name from Account]) {  
2      System.debug(a.Name);  
3  }
```

SOQL Returned Data



- Results are returned as a `List`
- Only fields that are selected are populated
 - Exception: The `ID` field is implicitly returned.
- Unpopulated fields will return `null`.
 - Attempts to reference fields not explicitly queried will result in a run-time exception
- When using Group By, results are returned as an `AggregateResult`

Writing SOQL in Apex

- There are two ways to embed SOQL into Apex code
- Square-bracketed expressions

```
List<Account> accounts = [SELECT Name FROM Account];
```

- String expressions

```
String s = 'SELECT Name FROM ACCOUNT';  
List<Account> accounts = Database.query(s);
```

Looping Through Query Results

There are three ways to retrieve database query results:

1. Select directly into a list (OK, but not great)

```
List<Account> accounts = [SELECT Name FROM Account];
```

2. Select directly into a loop and build a list (Very good)

```
List<Account> accounts = new List<Account>();  
for (Account a : [SELECT Name FROM Account]) {  
    accounts.add(a);  
}
```

3. Select into a List and iterate (Very good, required for batch Apex)

```
List<Account> accounts = new List<Account>();  
for (List<Account> acts: [SELECT Name FROM Account])  
{  
    for (Account a : acts) {  
        accounts.add(a);  
    }  
}
```

Walkthrough: Iterating through a List

Action Steps:

Use the Developer Console to execute anonymous

1. Write a query to select all Accounts, their related Opportunities and related Contacts and the name of the owner of the Account record
2. Select the results into a loop
3. Within the loop, iterate through each of the lists
4. Print each record to the debug log
5. Save the code in a text editor
6. Check the Debug Log within the Developer Console

Using Variables in SOQL

- Square-bracketed queries support binding to Apex variables using a colon (:)

```
String sIndustry = 'Utilities';  
List<Account> accounts = [SELECT Name  
                           FROM Account  
                           WHERE Industry = :sIndustry ];
```

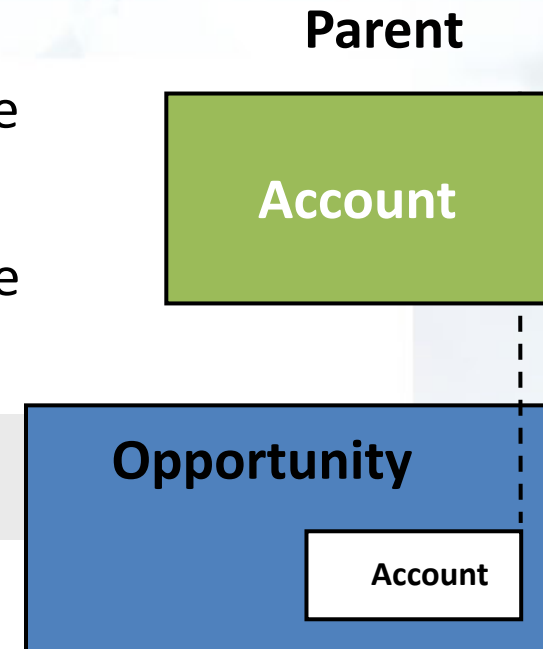
- String queries can be built dynamically

```
String sSOQL = 'SELECT Name FROM Account '  
sSOQL += 'WHERE Industry = \'Utilities\'';  
List<Account> accounts = Database.query(sSOQL);
```

Child to Parent in Apex

- Two member variables on the child refer to the parent.
 - The foreign key: the unique 18-digit ID of the related record.

```
Opportunity o = new Opportunity();  
o.AccountId = '001E000000HfF8E05S';
```



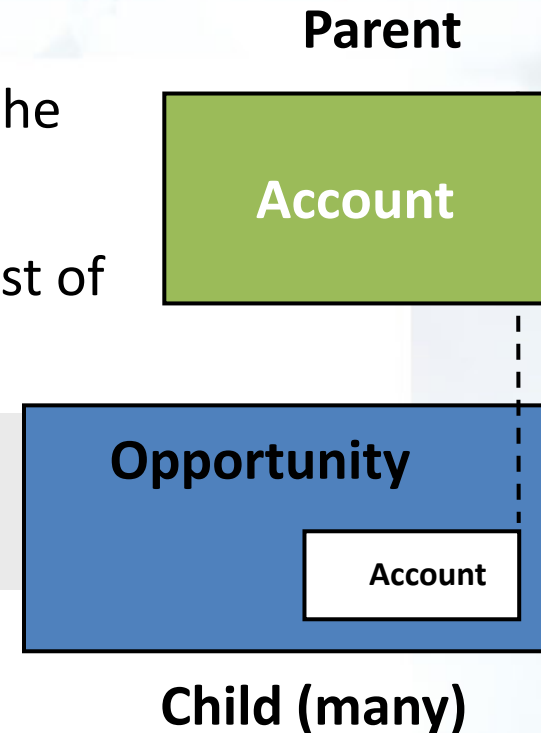
- The object reference: a reference to the remote sObject object.

```
Opportunity o2 = new Opportunity ();  
o2.Account = new Account();
```

Parent to Children in Apex

- One member variable on the parent refers to the children:
 - The related list reference: a reference to a List of related sObject objects.

```
Account a = new Account();  
a.Opportunities =  
    new List<Opportunity>();
```



- Data Manipulation Language
- The only way to modify data in Apex
- Available actions:
 - Insert
 - Update
 - Upsert
 - Actually performs both an Update and Insert
 - Generally should not be used if you are able to separate the new and existing records
 - Delete
 - Undelete
 - Merge (for select objects)

Invoking DML

- DML can be invoked in two ways

- DML Standalone statements:

```
Account a = new Account();  
a.Name = 'ABC Labs';  
insert a;
```

- Database methods:

```
Account a = new Account();  
a.Name = 'ABC Labs';  
Database.SaveResult sr = Database.insert(a);
```

- Both patterns can be used on either a single sObject or lists of sObjects

DML Standalone Statements



- Single Record Example:

```
Account a = new Account();  
a.Name = 'ABC Labs';  
insert a;
```

- If an error is encountered, a run-time exception is thrown

- Multiple Record Example:

```
List<Account> acts = [Select Name from Account];  
delete acts;
```

- If an error is encountered with one record, all records fail.
- A run-time exception is thrown

Database.SaveResult



- Returned with a `Database.insert()` or `Database.update()` method.
- There are similar `-Result` classes for the other operations
 - Example: `DeleteResult`
- If a single record is submitted, then a single `Database.SaveResult` object is returned
 - A result will always be returned on success or failure
- If a list of records was submitted, then a `List<Database.SaveResult>` list object is returned.
 - Each record in the result list corresponds to a record in the submitted list

Database Method Calls



- Single Record Example:

```
Account a = new Account();  
a.Name = 'ABC Labs';  
Database.SaveResult sr = Database.insert(a);
```

- If an error is encountered, no run-time exception is thrown
- The SaveResult is populated on success or failure

- Multiple Record Example:

```
List<Account> acts = [Select Name from Account];  
List<Database.SaveResult> srs = Database.update(acts);
```

- If an error is encountered with one record, all records fail.
- A list of SaveResults is populated

Database allOrNone



- When using the static Database methods, there is an optional Boolean parameter called `allOrNone`

- The value defaults to `true`.

- Example:

```
Database.insert(accounts);
```

- `allOrNone = true`

- Can be explicitly set to `false`

- Example:

```
Database.insert(accounts, false);
```

- If `false`, record errors are handled on a case-by-case basis, and the process continues despite having an error in a record.

SaveResult Methods

A `SaveResult` object has three methods available:

- `getId()`
 - Returns the ID of the record that caused the error
- `getErrors()`
 - Returns a List of `Database.Error` objects
- `isSuccess()`
 - Indicates whether or not the DML operation was successful

```
Account a1 = new Account(name = 'Bank One');
Account a2 = new Account(name = 'Bank Two');
List<Account> accounts = new List<Account> {a1, a2};
List<Database.SaveResult> srs = Database.insert(accounts, false);
for(Database.SaveResult sr:srs){
    if(!sr.isSuccess()){
        for(Database.Error e : sr.getErrors()) {
            System.Debug(e.getMessage());
        }
    }
}
```

sObject Relationships in Apex



- Relationship fields (Lookup & Master Detail) actually expose two interfaces in Apex
 1. The actual ID of the record being referenced
 2. An in-memory reference to the object of the related record

```
Contact c = [Select c.LastName
              from Contact c Where c.Id = '003E000000ECVTC'];
Account a = new Account(name = 'Team Sports');
insert a;
c.Account = a;
update c;
//the contact is not associated with Team Sports
a = new Account(name = 'Golf Team');
insert a;
c.AccountId = a.Id;
update c;
//the contact is now associated with Golf Team
```

DML and Loops

- Force.com has strict Governor Limits on the number of DML statements that can be executed within a single thread
- To avoid hitting Governor Limits when working with lists of records, move the DML statements outside of loops
- Example of improper design:

```
FOR(Account a :[SELECT id FROM Account]) {  
    Database.update(a);  
}
```

- Example of proper design:

```
List<Account> accounts = new List<Account>();  
FOR(Account a :[SELECT id FROM Account]) {  
    accounts.add(a);  
}  
Database.update(accounts);
```

DML and Batch



- Salesforce provides a technology called Batch Processing that allows for much higher governor limits
- Consider using Batch Processing when the volume of data needing to be processed exceeds the Governor Limit thresholds
- **NOTE:** Batch must be called asynchronously
- In order to take advantage of the higher Batch limits, DML statements must follow this design pattern:

```
for (List<Account> accounts: [SELECT Industry FROM Account])  
{  
    for (Account a : accounts) {  
        a.Industry = 'Construction';  
    }  
    update(accounts);  
}
```

Walkthrough: DML



Goal:

Update the opportunity close date for any overdue opportunity

Action Steps:

1. Write a query to select all open Opportunities with a close date in the past
2. Select the results into a loop
3. Within the loop, set the close date to today plus 30 days
4. At the end of the loop, update the Opportunities list
5. Execute the code and see that the dates for all the overdue opportunities have been changed

if / else Statements

- The only branching statements available in Apex
 - No CASE statement
- Pattern is `if`, `else`, `else if`

```
String color = 'Red';  
if(color == 'Green') {  
    System.Debug('Go');  
} else if (color == 'Yellow') {  
    System.Debug('Speed Up!');  
} else if (color == 'Red') {  
    System.Debug('Stop!');  
} else {  
    System.Debug('Invalid color');  
}
```

Classes



- Apex supports most object-oriented tenants through classes
- Classes support inheritance
- Classes support interfaces
- Classes can have properties and methods
- Classes can respect security settings
- Class methods can be overloaded
- Classes support different levels of external access
 - Private, Public, Global
- All Public classes within in a Salesforce namespace are automatically visible to all other classes within the namespace
 - No need to explicitly include file paths or headers in your code
- Classes are typically used as:
 - Unit Tests
 - Visualforce controllers
 - Lightning Component controllers
 - SOAP Webservices
 - REST Webservices
 - Email Services
 - Scheduled Batches

Class Access Modifiers

```
private | public | global  
[virtual | abstract | with sharing | without sharing | (none)]  
class ClassName [implements InterfaceNameList | (none)] [extends  
ClassOrInterfaceName | (none)] {  
    // The body of the class  
}
```

- **global**: This class may be accessible outside of your namespace.
 - Methods must be exposed either as **global** or as **webservice**
 - **global** methods not **webservice** are only accessible when installed as part of a managed package
- **public**: This class is visible across your application or namespace.
- **private**: This class is an inner class and is only accessible to the outer class or is a test class.

Class Inheritance

```
private | public | global  
[virtual | abstract | with sharing | without sharing | (none)]  
class ClassName [implements InterfaceNameList | (none)] [extends  
ClassName | (none)] {  
    // The body of the class  
}
```

- The `virtual` modifier declares that this class allows extensions
 - Classes not using the `virtual` modifier cannot be extended
- The `extends` keyword is used to extend a parent class and create a subclass.
 - The extending subclass can:
 - Extend the virtual class with new methods
 - Override existing methods in the virtual class
 - Access the original methods from parent class using the `super` keyword

Class Sharing



```
private | public | global  
[virtual | abstract | with sharing | without sharing | (none)]  
class ClassName [implements InterfaceNameList | (none)] [extends  
  ClassName | (none)] {  
  // The body of the class  
}
```

- Apex generally runs with System permissions
 - Triggers always run as System
- Classes have the ability to explicitly declare whether or not to run as System or as the invoking user
 - It is not possible to run code as or impersonate another user
 - `with sharing`: will respect all sharing and security settings of the invoking User
 - `without sharing`: will ignore all sharing and security settings and will run as system
 - No declaration: will inherit the context of whatever is calling the class
 - Example: When a trigger calls a class with no sharing declaration, the class will run as System since the trigger is running as System

Member Variables



- Member variables must be declared
- Member variables are strongly typed
 - `Var x;` – not valid
 - `String x;` - valid
- Support access modifiers
 - Declarations without an access modifier default to `private`
 - `private`: Accessible only within the class where it is defined
 - `protected`: Available to any inner classes or subclasses.
 - Can only be used by instance methods and member attributes
 - `public`: Can be called from anywhere in the namespace.
 - `global`: May be accessible by Apex outside of the namespace
- Values can be set during declaration

```
String s = 'Stony Point';  
Account a = new Account();
```

Class Properties

- Apex supports two design patterns for properties
 - Getters & Setters

```
String sFoo;  
  
public String getFoo() {  
    return sFoo;  
}  
  
public void setFoo(String s) {  
    sFoo = s;  
}
```

- Get; Set;

```
Public String foo {get;set;}
```


Class Properties in Lightning



- Class Properties are NOT supported in Apex classes used as controllers for Lightning Components
- Lightning Components cannot access class properties
- Design Pattern for Lightning Web Components
 - Apex Controller has methods
 - Lightning Controller imports methods into JavaScript
 - Lightning Controller calls methods and can pass parameters
 - Lightning Controller uses @wire to get results from method and updates Component
- Design Pattern for Aura
 - Component has an Attribute
 - Lightning Controller accesses Attribute with JavaScript
 - Apex Controller has methods
 - Lightning Controller calls methods and can pass parameters
 - Lightning Controller gets results from method and updates Component

Methods

- Can accept input parameters
- Support the same access modifiers as member variables
- Can return a value
 - Support void as a return type

- Can be static methods

```
public static Integer addIntegers(Integer x, Integer y) {  
    return x + y;  
}
```

- Can be instance methods

```
Public Integer addInts(Integer x, Integer y) {  
    return x + y;  
}
```

- Can be overloaded
 - Signature must be unique

Unit Tests

- 75% of all code must be covered by unit tests
- Every class doesn't need to be covered
- Every trigger must have at least 1% code coverage
- To test a trigger, write a test method that executes a DML statement

Walkthrough: Apex Controller



Goal:

Develop an Apex controller to retrieve all Opportunity records related to an Account

Action Steps:

1. Create an Apex class called OpportunityController
2. Make it public and with sharing
3. Create a public, static method called GetOpportunities that accepts an Account Id as a parameter
4. Write a query to select all Opportunities for the Account Id
5. Select the results into a List of Opportunities
6. Return the list

Walkthrough: oppCard Component



Goal:

Develop a Lightning Component to display an opportunity record in card format

Action Steps:

1. Create a Lightning Component called oppCard
2. Create a lightning card to display the Opportunity fields

Walkthrough: opportunityList Component



Goal:

Develop a Lightning Component to display a filtered list of Opportunities

Action Steps:

1. Create Lightning Component called opportunityList
2. Create an iteration to loop through the opportunities and display them on the <c-opp-card> component
3. Build a picklist to allow the user to filter the display to only certain opportunities

Survey

Please take this time to fill out the course survey.

<http://surveys.stonyp.com>

Your instructor will provide you with the class code to use.

Question and Answer



Please take this time to ask the instructor any questions you may have about the topics covered in this class.

For additional training, please visit www.stonyp.com.