* P1 was assigned today
  + No standard controllers
  + Test any and all classes you create
  + Live demo: deploy some part of the project
    - Don’t have to deploy experience cloud site
    - You can for bonus points
  + 10 minute presentation
    - Don’t show code
* Upsert data
  + Combination of insert and update
  + Matches using id if it already exists
  + If an id is found multiple times, it will throw an error
  + Helps prevent redundant data
  + Several tools we can use
    - Data loader
    - CSV
  + External IDs
    - Used for migrations and integrations
    - Helps the auditing of information across both sites
    - Always indexed and treated as an id field
    - One object can have 25 external id fields
  + External id or not, salesforce will always give it a salesforce id
  + Object relationships
    - Parent objects have to be uploaded before the child object
    - This is because of required fields
    - There is a correct order to upload them in
    - Relationships are expressed through related lists and lookup ids
  + Configure upsert action to traverse object relationships
  + Need to know IDs before upserting
  + Example
    - Graphical user interface, text, application

      Description automatically generated
  + Always start with users object
  + Tools for data management
    - Graphical user interface, application

      Description automatically generated
  + Dataloader
    - Can load any object supported by the API
    - Load 5,000,000 records
    - Can schedule regular data loads
    - Export data for backup
    - Mass delete supported objects
    - We will use dataloader.io instead since it is easier to work with
    - Fully supported
    - Import and export CSV
    - Can load to or out of database with JDBC
    - Supports custom relationships for upsert
    - Can be run through the command line
      * Gives some extra features
    - Batch mode
    - Getting it
      * System administrators can download it
      * Can also interact with the browser
    - A picture containing table

      Description automatically generated
  + Data import wizard comes down to ease of use
    - It has fewer functionalities
* Bulk API
  + Used for really large batch sizes
  + High volume data loads
  + Works asynchronously
  + High stability
  + Will not run until there are resources available
  + Better control when doing lots of data
  + Only from single large CSV
  + Uses a queue to process in smaller jobs
  + Saves results of each job and returns it to a user
  + Data loader SOAP API by default
    - This can be changed to use Bulk API in the settings
    - You can also configure series or parallel if you need to ensure data security or if you run into issues
  + You can monitor this in the bulk data load jobs in setup
* Visualforce
  + Page-centric UI framework
  + Allows us to build complex UIs that are native to salesforce
  + It is a markup-based language
    - Uses tags
    - Similar to html
  + You can use HTML directly inside your visualforce page
  + Uses server-side standard controllers to provide easy integration and interaction with the salesforce database
  + Salesforce servers convert the framework specific tags over to HTML
    - Then the browser can read it
  + Every visualforce page will have the <apexpage> tag starting the page
  + The <apexpage> has an attribute called renderAs which you can give the value of pdf to have your page render as a pdf
    - This makes our pages printer friendly
    - However, it cannot run any JavaScript
  + Controllers
    - What allow us to interact with our data/the server
    - 3.5 types
      * Standard
      * Standard set/list
      * Custom
      * Controller extension
    - Expression syntax
      * {!}
      * Tells the system that the following value is not to be taken literally
      * Instead, a reference to another value
    - Action binding
      * Coupling or binding a method from the controller to an event on the page
      * Think button bonded to an action on the page
    - Data binding
      * Coupling or binding of a variable from the controller to the page and vise versa
      * This is 2 way binding
        + Modifying on one are reflected on another
    - Standard controller
      * Set of prewritten salesforce-made code that provides functionality for our visualforce pages
      * They are automatically created for each standard and custom object
      * Contain methods that perform actions like saving or deleting a record
      * Designed to mimic the functionality of record-detail pages in the lightning experience
      * To declare one
        + <apex:page standardController=”Account”>
      * Can see modify and work with the records in the account object
      * Visualforce gets the record ID for which record its working on from the URL
      * Cancel, save, quicksave and delete actions
    - Standard set/list controller
      * Allow you to see groups of records at the same time
      * Only available for
        + Account
        + Asset
        + Campaign
        + Case
        + Contact
        + Contract
        + Idea
        + Lead
        + Opportunity
        + Order
        + Product2
        + Solution
        + User
        + ALL CUSTOM OBJECTS
      * To use
        + <apex:page standardController=”Account” recordSetVar=”myVar”>
      * The list view you most recently saw will filter your results
      * The records can be displayed in a table
        + The records are paginated or grouped in sets of 20
        + This controller lets us navigate natively among pages
      * Same commands as before except delete
      * To reference the list in a table, value=”myVar” var=”currentRecord”
    - Extensions
      * You can extend the functionality of an existing controller
      * We can only have one controller per page
      * You can use the prebuilt stuff as well as add additional functionality
      * To do this, your class must have a constructor that takes in the controller it’s extending as a parameter
      * You can have as many extensions as you want but if two controllers have naming conflicts of methods or variables inside of them, the one defined first takes precedence
      * Use the extensions attribute to specify which extension to use
    - Custom controllers
      * Controllers completely made by you
      * They are done by creating an apex class
      * Reference the class name as the controller=
      * You must declare all variables you want to use
      * You must write all methods for retrieving records etc
      * There are very specific naming conventions you HAVE to follow
        + If you want to be able to reference a variable on the page, you need to have getters and setters
        + The variables and their gets/setters need to be nameOfVariable, setNameOfVar, getNameOfVar

Definitely use camelCase

* + - * + Need to allow visualforce pages access to our methods and variables

They must be public

* + When to use which controller
    - When you only need the functionality of a standard or list controller, use them
    - If you need the functionality of a standard with additional functionality, use an extension
    - If you need to show records despite permissions and security, you can use a custom controller
    - If you need none of the functionality of a standard controller, you can use a custom controller
* Organizing your visualforce pages
  + Apex page block
    - Basically a div from HTML
    - Also includes some styling similar to that of a record detail page in lightning experience
    - You can also set the title attribute to add a header to a block
  + Apex page block section
    - Must be a direct child of apex page block
    - Creates a new section of the page block
    - Also has nice features when paired with other components
  + Apex page block buttons
    - Must also be a direct child of the apex page block
    - Can hold apex command buttons inside of it
    - According to the documentation, the buttons will add styling to the command buttons, but you won’t see it
    - It will put the buttons both at the top and bottom of the page block
* Accepting input with standard input components
  + Every input component must be inside of an apexform component
  + Apex input component
    - Does not include any salesforce styling
    - Not bound to any field on any salesforce object
    - The value html-5.0 must be passed into the doctype attribute of the opening apexpage component
  + Apex checkbox
    - Checkbox that has no associate with a salesforce record field
  + Apex input text
    - Text value that is not bound to a field on a specific salesforce record
  + Apex input text area
    - Text area that is not bound to a field on a specific salesforce record
  + Apex input field
    - Will allow you to edit a field on a record without needing a custom controller or extension
  + Validation rules are automatically applied if you use a standard controller when providing input
    - If a validation rule displays at the top of the page, we need to use more visualforce components
    - <apex:messages? And <apex:pageMessages> will allow us to display our error messages on the page
    - If you have a custom controller or extension, use a try/catch to catch errors and use the apexPages.addMessages() to add the exception to the page
* Displaying data with standard components
  + Apex output
    - Allows you to display a field value of a salesforce record by specifying its value
    - Is read only
    - Does not have to be placed in an input form tag
  + Apex output label
    - Commonly paired with non-apex output fields as well as input components
    - Lets us give our elements a label
    - Gets associate with input/aputput field by specifying the id of the element we want to label to the for attribute of the component
  + Apex output link
    - Basically a <a>
  + Apex output panel
    - Useful when paired with apex:actionSupport
  + Apex output text
    - Allows us to display text to our user
* Displaying records in visualforce tables
  + Apex page block table
    - Must be inside an apex page block or block section
    - Will include salesforce styping
  + Apex data table
  + Both can display up to 10,000 records
  + Can have up to 1,000 records that can be edited
* Inline editing
  + Editing values while displaying them
  + <apex:inlineEditSupport>
  + Must be a direct child of datalist, form, outputfield, pageBlock, pageBlockSection, pageBlockTable or repeat
  + Easiest way is in an apex output field
  + All of this has to be wrapped in an apex form
  + You can also do the apex detail component
* Partial page rerendering
  + When you visit a visualforce page the browser makes a request to the visualforce servers with a converted url, finds the correct page, parses any specific markup to html and returns it to the browser
  + Our controllers remain on the server
  + To interact with anything in our controllers we have to make a call back to the server
    - This is computationally wasteful
    - Same for when we submit
  + For more selective rerendering, we can wrap any section that the server should process within an <apex:actionRegion> component
    - When a DOM event occurs in that region, the server will only process events in that region
  + The rest of the page is returned to the browser in the same state it was before the event fired
  + To do this asynchronously, we can use <apex:actionSupport>
    - The event attribute holds the name of the DOM event like “onClick”
    - reRender holds a comma-separated list of id values that are to be refreshed when the event occurs
* View state
  + Hidden data describing the state (size, type, name, value) of our components on the page
  + This data is what is sent to the server whenever a call is made to describe the current state of the page
  + It allows for persistence of values on the page across server requests
  + It has a size limit of 170 KB
  + The response time will slow if the view state gets large
  + Decreasing size
    - Only use apex form and input fields when absolutely needed
    - Rework your SOQL queries so that you are removing unneeded records
    - Make user of the transient keyword on your variables in your controllers and extensions
  + Static and transient variables are not included in the view state
  + Transient variables are instance variables that are not saved and are remade with each page refresh