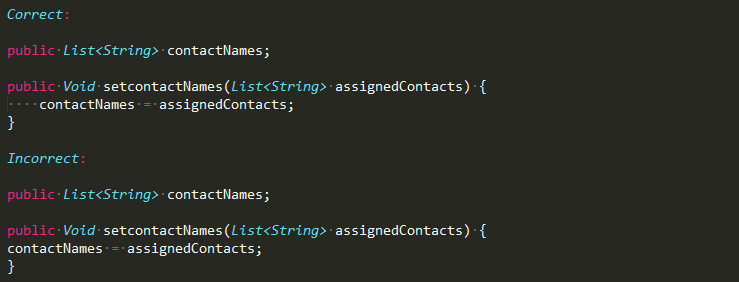
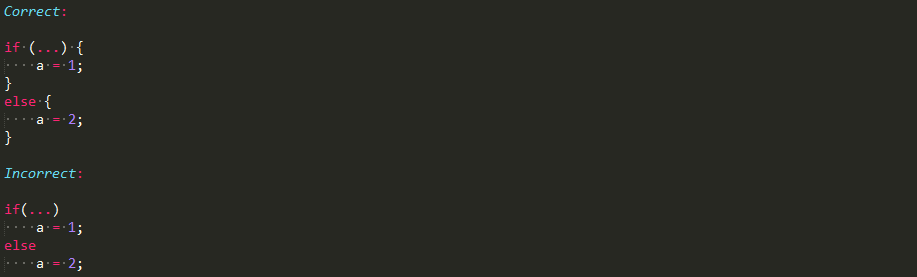
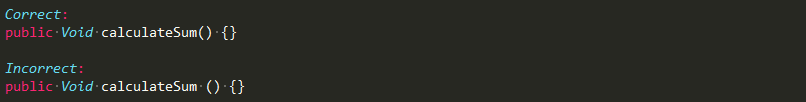
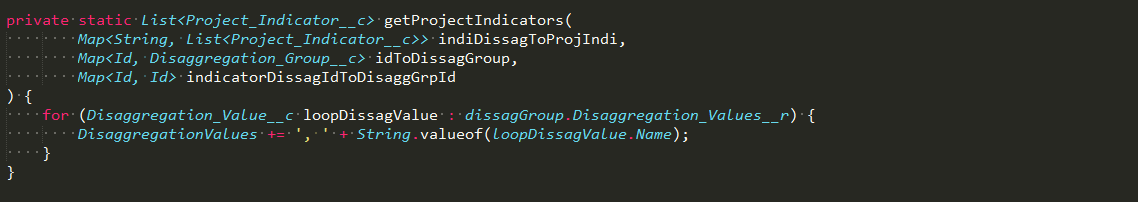
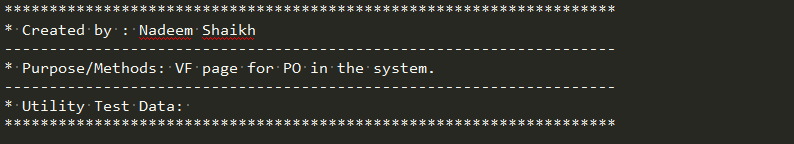
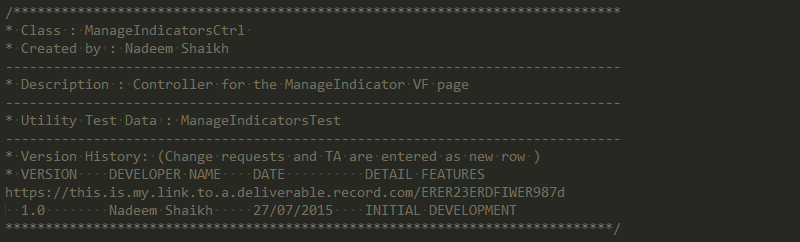
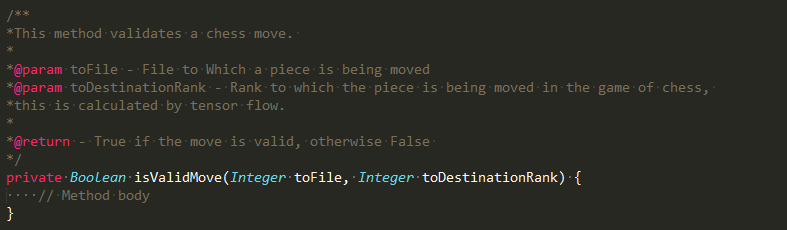
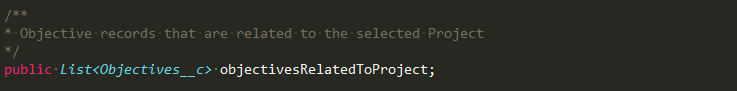
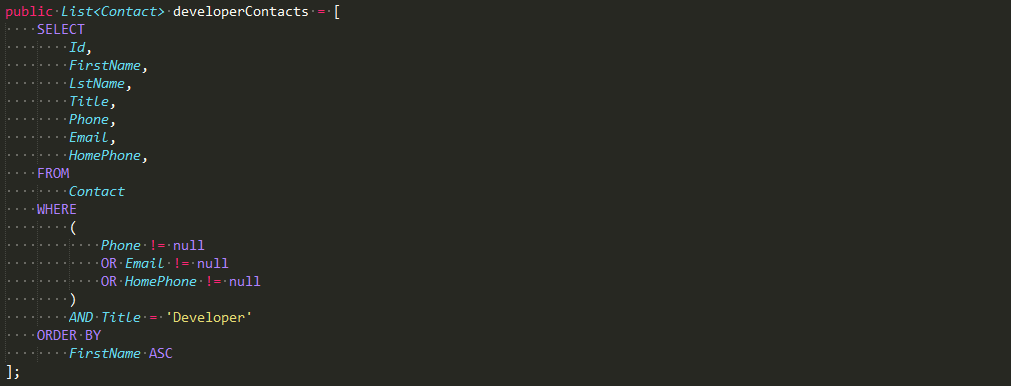
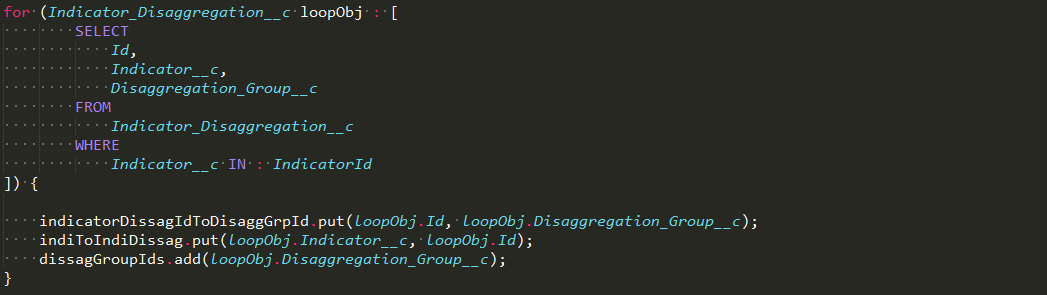
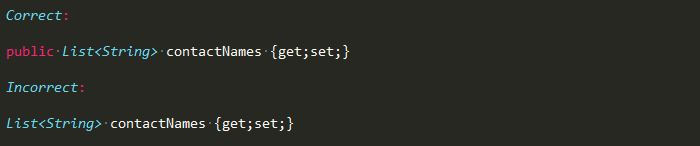
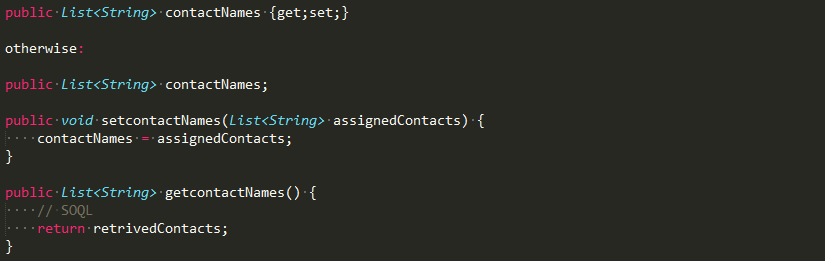
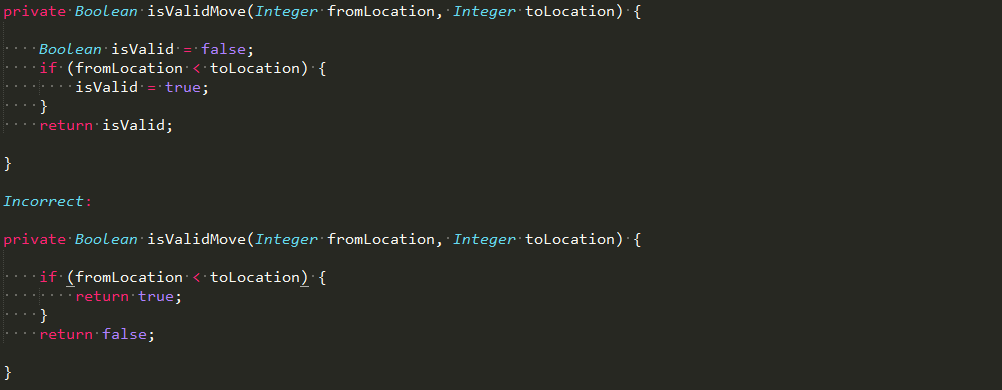
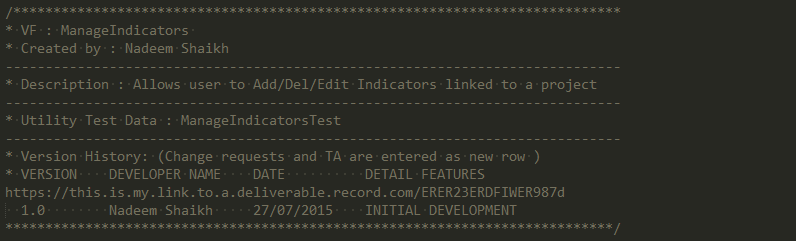
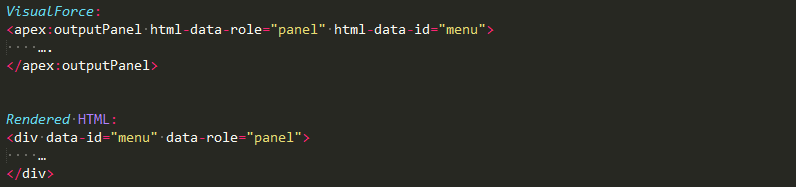
1. **Code Layout and Commenting**
   1. Use spaces for code indentation.
   2. Tab width should be 4 whitespaces long.
   3. Line length should be max 100 characters; 80 characters are ideal.
   4. Keep method definition short, ideally 80 lines. If the method definition increases the ideal number of lines try to break it into multiple methods that support each other.
   5. Try to make methods context inspecific.
   6. Code block that has the same level of depth should have same level of indentation:
   7. Do not use underscores “\_” in variable and/or method names to separate two words.
   8. Add a space before the starting curly bracket “{“ of a code block; Constructs such as if, else, for, while, do, switch, methods, classes, getter, setter count as code blocks. Example:
   9. Always include curly braces {} to indicate the start and end of looping and branching constructs:
   10. Do not add spaces between method names and parentheses:
   11. Break methods with longer definition/calls in the following manner
       1. Note the indentation is one level deep for parameters on the new line, this is to add visible difference between the method in parameters and statements in the method definition.
   12. Remove all trailing spaces at end of a line.
   13. Group method by their access specifier so All public methods are together, privates are together and so on.
   14. Follow method hierarchy i.e. method definition and invocation should be in close proximity to each other as much as possible after following scope grouping.
   15. Define wrapper classes at the bottom.
   16. Use javadoc style of code commenting as detailed in the following sections.
   17. Data types should be written using Pascal Casing:
       1. String
       2. List
       3. Set
       4. Void
       5. PageReference
   18. Access specifiers should be written using Lower Camel Casing:
       1. public
       2. static
       3. final
       4. protected
       5. private
       6. with sharing
   19. Inheritance should be Pascal Casing:
       1. Implements
       2. Extends, etc.
   20. Final Variables are defined using all capital letters with underscores separating two words.
   21. Variable names should be:
       1. Human readable
       2. Descriptive
       3. Should not require knowledge of the code base to understand is purpose/use
       4. Should not have its data type prefixed to its name.
       5. Exception to this rule can be added for naming variables in looping constructs such as for, foreach, while, etc where naming the variable as i,j,k for example is more productive and readable rather than figuring out a depictive variable name.
   22. Code commenting:
       1. All classes, triggers and visualforce pages should begin with an info block:
          1. Info block for projects with version control:
          2. Info block for projects without version control:
          3. Comments should be maximum of 100 characters per line
          4. Info block contains the type of artifact Class/Trigger/VisualForce.
          5. Created by
          6. Description of the class and name of its test class.
          7. *Change history, which starts with link to the deliverable/card the PM tool which contains the technical/functional specification and requirements. Required only if no version control tool such as Git, SVN is maintained.*
          8. *Followed by the deliverable include a brief information about who made the change and when. Required only if no version control tool such as Git, SVN is maintained.*
       2. Method level commenting:
          1. *Required only if the method name, signature and scope doesn’t provide enough information on purpose of the method.*
          2. All method should have method level commenting in Javadoc format, ex:
          3. Comments should be a maximum of 100 characters per line
          4. The comment starts with /\*\* and ends with \*/
          5. Each new line begins with a \*
          6. If a method is parameterized use the @param annotation describe each accepted parameters of the method.
          7. Each method parameters should be defined on a new line one below the other.
          8. Use the @return notation to describe the why and what of the return type of the method.
       3. Variable level commenting:
          1. *Required only when the variable name and scope is not sufficient to explain the purpose of the variable.*
          2. Variable level commenting should follow the Javadoc standard:
          3. Comments should be a maximum of 100 characters per line
          4. In most cases the variable should be descriptive enough to give away its use if that is not possible add comments to describe its purpose.
          5. The comment starts with /\*\* and ends with \*/
          6. Each new line begins with a \*
          7. Short description of the purpose of the variable. Refrain from using the type of the variable in its description example: A list of Objectives, Boolean to store if data is valid or invalid use something like the one used in the code snippet.
   23. Formatting SOQL queries:
       1. Keywords should be all capital letters.
       2. Field names should respect case sensitivity of field API names.
       3. Object names should respect case sensitivity of object API names.
       4. Relationship names should respect case sensitivity of relationship API names.
       5. SOQL query longer than 100 characters should be formatted in the following manner:  
          1. SELECT, FROM, WHERE and ORDER BY clauses (but not limited to these clauses) are written on a new line and have the same level of indentations.
          2. Keywords are written in all capital case.
          3. SOQL queries follow the the APEX indentation model where SOQL query blocks at the same level has the same level of indentation.
          4. Each field in each clause is written on a different line, have a look at the SELECT, WHERE and ORDER BY clause in the example above.
          5. Logical operator always precedes field names.
          6. Use parenthesis () to group complicated logical operations.
          7. Subqueries follow the same SQOL layout pattern just one level deeper in the indentation hierarchy.
       6. SOQL queries written in for and if statements should be formatted in the following manner:  
          1. SOQL queries start one level deep in the indentation hierarchy compared to the for loop/if statement
          2. Closing parenthesis is placed on a new line at the same indentation hierarchy of for loop/if condition
   24. Do not leave commented code in APEX classes or triggers. Comments should only be used to describe functionality that is not obvious by the method structure and signature and add information about the input and return parameters.
   25. Custom labels are defined using Camel Casing.

1. **APEX**
   1. APEX class names should not be over 35 characters.
   2. Domain classes should be named in the following manner : ObjectName\_Domain
   3. Follow Pascal casing also known as Upper Camel Case for naming:
      1. replace.pngAPEX Classes
   4. Follow Lower Camel Case for naming:
      1. replace.pngVariable names
      2. replace.pngMethod names
   5. replace.pngFinal Variables are defined using all capital letters.
   6. Bulkification:
      1. No SOQL or SOSL queries inside loops.
      2. No DML statements inside loops.
      3. Do not make calls to asynchronous apex (@future, queues, batches) inside loops.
      4. Avoid making describe calls inside loops.
   7. Always use the with sharing keyword in all apex classes. This can only be bypassed by presenting a strong case by the technical team for creating an APEX class without the “with sharing” keyword.
   8. All synchronous APEX should be Bulkified:
      1. Methods should take in List, Set, Map of an object instead of a single object.
      2. Methods should return List, Set, Map of an object instead of a single object.
   9. Use Asynchronous APEX for logic that does not need to be executed synchronously i.e. not in real time such a need might arise during sharing of records, callouts to third party web services, and processing large amount of data, etc use:
      1. @Future
      2. Queues
      3. Batch class
      4. Scheduled class
   10. All asynchronous APEX should be Bulkified.
   11. System.debug is a resource intensive task, use it only when the debug logs do not provide information about the code block being executed.
   12. Prevent SOQL and SOSL injection attacks by using one of the following:
       1. Static queries
       2. Binding variables or
       3. Applying the escapeSingleQuotes method on variables received as input from user.
   13. When querying large number of records, use the SOQL query in foreach loop. Example [here, refer to point no #6](https://developer.salesforce.com/page/Apex_Code_Best_Practices).
   14. Use SOSL over SOQL where possible - it's much faster.
   15. Use Dynamic SOQL queries for managed package.
   16. Apex classes should provide exception handling. Exception control should be added for exceptions which cannot be predicted or evaluated for during runtime for example:
       1. Serializing a JSON string to an object.
       2. Issuing a describe call for an object whose name is obtained by user input.
   17. Optimize SOQL query for performance by:
       1. Using:
          1. WHERE
          2. IN
          3. EQUAL TO
          4. LIMIT X
       2. Avoiding:
          1. NOT EQUAL TO
          2. NOT IN
   18. Always include access specifier for all class properties and methods:
   19. All properties within a class should have their getter and setter defined either inline or explicitly as shown below:
   20. Do not hardcoded references to Ids, Record type names, Picklist values use Custom labels instead.
   21. Always use Developer names when referencing record types.
   22. Use describe calls to retrieve the record type ids; save the record type label name in a custom label.
   23. Variables that shouldn’t be part of the view state should be declared transient in order to conserve view state.
   24. Use APEX limits class to pre-empt hitting governor limit exceptions; for example:
       1. Check for number of DMLs available before performing a DML on a list
       2. Check for number of SOQL queries available before performing an SOQL query
   25. Avoid using multiple return statements:



1. **Triggers**
   1. Trigger names cannot be over 255 characters.
   2. Triggers follow all the layout, naming, and coding conventions specified in the Apex section of this document.
   3. All triggers and associated handler class should be bulkified.
   4. All triggers should have recursion protection.
   5. Create one trigger per object per event i.e. each object will have 7 triggers.
   6. Triggers should adhere to the following naming convention:
      1. Triggers name should begin with a prefix “Trg\_”
      2. Trigger name should contain the prefix and object name followed by an \_ and the event name, example : Trg\_Contact\_AfterInsert
   7. Triggers are logic less hence they do not contain any business logic. A trigger just delegates the processing to the appropriate handler class which performs all the business logic.
   8. At max a trigger can contain routing logic that routes records to respective business logic. To avoid recursion within trigger, maintain a map of records that are already processed/routed.
   9. Define one handler class per object.
   10. Handler class name should be the name of the object concatenated with the word “Handler”. For a trigger on Contact the handler class will be named : Contact\_Handler
   11. Handler classes should be context in-specific i.e. for example AfterInsert, AfterUndelete and AfterUpdate trigger should in most cases be able to call the same method in the handler class.
   12. Always ask can/should this be built using process builder, queued, @futured or batched before adding a functionality in a trigger.
2. **VisualForce**
   1. VisualForce page names should be in Pascal Case (Upper Camel Case).
   2. All VisualForce pages should begin with an info block:



* 1. Set applyBodyTag=”False” and applyHtmlTag=”False” and create the <html>, <head> and <body> tag yourself.
  2. Always include the doctype="html-5.0" property in the <apex:page> tag
  3. Always include the title=”” property in the <apex:page> tag.
  4. Add CSS resources at the top of the page in the <head> section.
  5. Add JavaScript resources at the bottom of the page tag.
  6. Make use of CDNs for standard libraries such as jQuery, Bootstrap, Google fonts, etc. as much as you can.
  7. Static resource names are defined using Pascal Casing, unless the file is a part of an external plugin or library. Example: CommonStyle.css
  8. Use jQuery over plain JavaScript.
  9. Do not use a mix of jQuery and JavaScript stick to one and use it consistently.
  10. Use j$ = jQuery.noConflict() method to partition references to other libraries using the $ notation.
  11. Substitute VisualForce components with standard HTML tags as much as you can for example <apex:dataTable>, <apex:pageblockTable> with HTML <table> as these are faster to load and consume less view state.
  12. All hardcoded text should be replaced with custom labels to support quick and easy modification, and localization using translation workbench.
  13. Do not leave commented tags in Visualforce pages.
  14. Add Pass-through attributes to VisualForce components to better support jQuery, ng, React etc. Example:

1. **Unit Testing**
   1. Test class names cannot be over 40 characters.
   2. Test classes follow all the layout, naming, and coding conventions specified in the Apex section of this document.
   3. Test classes should also have the info block which is described in the APEX section of this document.
   4. Each test class should be a private class and should have the @isTest notation.
   5. All APEX classes should have its own test class.
   6. The name of the test class should contain the name of the APEX class it is associated with ending with a postfix “\_Test”.
   7. Create your own test data for use in test cases at all times.
   8. Do not use “SeeData=True” in test classes, unless you need to deploy triggers/apex class/controllers using the MetaData API directly into production via code.
   9. Test code that is actually testing for functionality should start with Test.startTest() and end with Test.stopTest(). Creation of test data and the ground work should happen before the Test.startTest() method is called in the test class.
   10. Use assets statements to check the intended vs actual result.
   11. Always write test cases to cover:
       1. Single records
       2. Bulk records, test with a minimum of 200 records
       3. Test for positives by using System.assert() or System. assertEquals()
       4. Test for negatives by using System.assertNotEquals()
       5. 95% code coverage
   12. Use the [MockHttpResponseGenerator class to simulate responses to requests to outbound web services made via apex callouts](https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_restful_http_testing_httpcalloutmock.htm).
   13. Run test for specific test classes that are being written and then run all test in the org to ensure all test cases run without errors in unison.
   14. Always use test helper to create test data, the helper classes should be split into two (but not limited to) reuseable parts:
       1. Creation of records
       2. Insertion of records
   15. Run the Force.com Security Source Scanner to test your Org for a number of security and code quality issues such as cross site scripting, access control issues, frame spoofing, etc.
   16. Optimize test classes by running DMLs on collections.
   17. Most test cases do not need records to be inserted if the code is partitioned correctly; one can pass the list of (not inserted) records to the handler methods, leverage this technique wherever possible.
   18. Method names in test classes should follow the method naming convention as described in the APEX section of this document.
2. **Configuration**
   1. If multiple team members needs access to an org use an emailing list as the user email address example: team@something.com or devs@something.com.
   2. If such an email address is not available add a forwarding rule to relay password reset, access token, and security token emails to all the users in the team who would need access to the org.
   3. Under password policy set the password to never expire, this should not be done on a live system without client authorization and assessing client access control policies; Only on a Sandbox or a DE org.
   4. Always deploy profiles while deploying from Sandbox to Production.
   5. Changeset names should be versioned for example : ManageIndicators VF v1
   6. Add links to deliverable/trello card/Jira ticket in the description section of a Changeset.
   7. All changes should be made in the Sandbox and then deployed to production this helps maintain both the system consistent.
   8. Always do an impact analysis i.e. evaluate Triggers, Classes,Test classes and automation such as workflow rules and validation rules before converting a lookup relationship to master detail as it may cause runtime exception and/or assert statements to fail during test execution.
3. **Version control**
   1. Maintain separate directories for IDE and Version control
   2. Always create a branch from the develop branch
   3. Add a 2 at max 3 character prefix to the branch name of your initials followed by an \_ and then the feature name/number being worked upon
   4. Commit your code as often as you can
   5. Every commit should include a quick indication of what the commit is for and a small summary of:
      1. What has changed?
      2. Why it has changed?
      3. What it has changed to?
   6. A feature is submitted for code review by creating a pull request and assigning it to the reviewer (Owen for the time being)
   7. Pull request submissions also follows the commit summary and description pattern, with an addition of including
      1. Trello card/JIRA ticket/Deliverable link
      2. File names part of the feature in the following order:
         1. Added denoted by A: FileName
         2. Edited denoted by E: FileName
         3. Deleted denoted by D: FileName
   8. Always pull the latest snapshot of the repo before the code review is performed
   9. All changes that arise after the code review are committed to the same branch and the reviewer is notified via adding a comment on the pull request (this process will change in future)