Salesforce Certified Platform Developer I

Salesforce PDI

Version Demo

Total Demo Questions: 46

Total Premium Questions: 466

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Salesforce Fundamentals	153
Topic 2, Process Automation and Logic	115
Topic 3, User Interface	104
Topic 4, Testing, Debugging, and Deployment	93
Topic 5, Mix Questions	1
Total	466

QUESTION NO: 1

A developer needs to display a translated, organization-wide error message in an Apex class. The message must be maintained declaratively and must not be hard-coded in Apex.

What should the developer use?

A. A custom label

B. A custom setting hierarchy record

C. A static resource

D. An Apex constant

ANSWER: A

Explanation:

A custom label is the appropriate solution. Custom labels store text values that can be translated and referenced from Apex, Visualforce, Aura components, Lightning web components, and other Salesforce features. In Apex, a developer can reference a label by using syntax such as `System.Label.Order_Error_Message`.

Because labels are metadata, an administrator can maintain the displayed message without modifying the Apex class. Translation Workbench can provide translated values where enabled. See Access Custom Labels in Apex.

QUESTION NO: 2

What can used to delete components from production?

A. A change set deployment with the delete option checked

B. An ant migration tool deployment with destructivechanges xml file and the components to delete in the package .xml file

C. A change set deployment with a destructivechanges XML file

D. An Ant Migration Tool deployment with a destructiveChanges.xml file that lists components to delete and a package.xml file containing only the API version.

ANSWER: D

Explanation:

An Ant Migration Tool deployment with a `destructiveChanges.xml` file that lists components to delete and a `package.xml` file containing only the API version is the supported Metadata API pattern for deleting metadata from a production org. The deployment is performed through the Metadata API, which the Ant Migration Tool uses. The destructive manifest identifies the metadata types and members targeted for removal, while the package manifest supplies the required API-version information without needing to deploy source components. For example, the destructive manifest can name an Apex class, custom object, field, or other supported metadata component, using the applicable metadata type and member name. Salesforce evaluates the destructive changes as part of the deployment and applies the deletion when the deployment succeeds. Before running this type of deployment in production, teams should validate it and ensure that dependencies have been handled, because a component cannot be deleted while metadata that depends on it remains. Salesforce documents destructive changes as a Metadata API deployment capability and describes the required deployment-manifest structure. See Delete Components from an Org and Ant Migration Tool Guide.

QUESTION NO: 3

Flow Builder uses an Apex action to provide additional information about multiple Contacts, stored in a custom class ContactInfo.

Which is the correct definition of the Apex method that gets the additional information?

A. @InvocableMethod(label= ' Additional Info ') public ContactInfo getInfo(Id contactId) { /* implementation */ }

B. @InvocableMethod(label= ' Additional Info ') public static List < ContactInfo > getInfo(List < Id > contactIds) { /* implementation */ }

C. @InvocableMethod(label= ' Additional Info ') public static ContactInfo getInfo(Id contactId) { /* implementation */ }

D. @InvocableMethod(label= ' Additional Info ') public List < ContactInfo > getInfo(List < Id > contactIds) { /* implementation */ }

ANSWER: B

Explanation:

`@InvocableMethod(label= ' Additional Info ' ) public static List < ContactInfo > getInfo(List < Id > contactIds) { /* implementation */ }` is the appropriate definition because an Apex method made available as a Flow action must be declared `static` and annotated with `@InvocableMethod`. The annotation exposes the method to Flow Builder and its label supplies the action name presented to flow authors.

The method accepts one collection parameter, `List<Id>`, which lets Flow pass the IDs for multiple Contact records in a single invocation. This collection-oriented contract is essential for Apex actions used by Flow because Flow can process record collections and Apex must be designed to handle the invocation in bulk rather than one Contact at a time. The returned `List<ContactInfo>` similarly provides a collection of result objects for Flow to consume. When `ContactInfo` is used as an Apex-defined output type, its fields intended for Flow should be exposed with `@InvocableVariable`.

Salesforce documents the requirements and supported parameter/return patterns for invocable methods in the @InvocableMethod reference. Details for exposing members of an Apex-defined type are available in the @InvocableVariable reference.

QUESTION NO: 4

what are the three languages used in the visualforce page?

A. Javascript, CSS, HTML

B. Apex, Json, SQL

C. C++, CSS, query

ANSWER: A

Explanation:

JavaScript, CSS, HTML is correct because Visualforce pages are web pages that are rendered for use in a browser and can incorporate the standard front-end technologies used to structure, style, and add behavior to a user interface. HTML supplies the page's rendered structure and content, CSS controls presentation such as layout, colors, spacing, and responsive styling, and JavaScript provides client-side behavior such as event handling and dynamic updates. Visualforce also supplies Salesforce-specific markup components and can use an Apex controller to obtain data and implement server-side logic, but those capabilities complement the page's browser-facing technologies rather than replacing them. In practical Visualforce development, a developer can use standard HTML elements, include CSS through static resources or stylesheets, and use JavaScript directly or through static resources while respecting Salesforce security and platform guidance. Salesforce describes Visualforce as a framework that lets developers build custom user interfaces hosted on the Lightning Platform, with pages that use markup and can be enhanced using familiar web-development techniques. See the Salesforce Visualforce Developer Guide introduction and the Salesforce static resources documentation for supported approaches to supplying JavaScript and CSS assets to Salesforce user interfaces.

QUESTION NO: 5

A developer must implement a CheckPaymentProcessor class that provides check processing payment capabilities that adhere to what is defined for payments in the PaymentProcessor interface.

apex

Copy

public interface PaymentProcessor {

void pay(Decimal amount);

}

Which is the correct implementation to use the PaymentProcessor interface class?

A. public class CheckPaymentProcessor implements PaymentProcessor { public void pay(Decimal amount) {} }

B. public class CheckPaymentProcessor implements PaymentProcessor { public void pay(Decimal amount); }

C. public class CheckPaymentProcessor extends PaymentProcessor { public void pay(Decimal amount); }

D. public class CheckPaymentProcessor extends PaymentProcessor { public void pay(Decimal amount) {} }

ANSWER: A

Explanation:

`public class CheckPaymentProcessor implements PaymentProcessor { public void pay(Decimal amount) {} }` is the valid implementation because Apex uses the `implements` keyword when a concrete class agrees to fulfill an interface contract. The `PaymentProcessor` interface declares a `pay` operation that accepts one `Decimal` parameter and returns no value. Therefore, `CheckPaymentProcessor` supplies a method with that same name, parameter type, and return type. Its `public` visibility is appropriate for the interface contract, and the braces provide the required executable method body. Although the body is empty in this example, it is syntactically a concrete implementation; production code would place the check-payment processing behavior inside it. This design allows application code to depend on the `PaymentProcessor` abstraction while using a check-specific implementation wherever payment processing is needed. Salesforce documents that an interface specifies method signatures, and a class implements the interface by providing the required method implementations. See the Salesforce Apex Interfaces documentation and the Apex object-oriented programming Trailhead material.

QUESTION NO: 6

When importing and exporting data into Salesforce, which two statements are true?

Choose 2 answers

A. Bulk API can be used to import large data volumes in development environments without bypassing the storage limits.

B. Bulk API can be used to bypass the storage limits when importing large data volumes in development environments.

C. Developer and Developer Pro sandboxes have different storage limits.

D. Data import wizard is a client application provided by Salesforce.

ANSWER: B C

Explanation:

Bulk API can be used to bypass the storage limits when importing large data volumes in development environments. This is a useful capability for development and test data loading: Salesforce permits Bulk API loads in Developer and Developer Pro sandboxes without enforcing the sandbox storage allocation during the load. It enables developers to work with realistic, high-volume data sets for testing integrations, batch processes, and custom code. This behavior does not change the general purpose of Bulk API, which is Salesforce's asynchronous interface for efficiently processing large sets of records. Salesforce documents Bulk API and its use for loading or deleting large volumes of data at Bulk API and Bulk API 2.0 Developer Guide.

Developer and Developer Pro sandboxes have different storage limits. A Developer sandbox includes a smaller data-storage allocation, while a Developer Pro sandbox provides a larger allocation intended for development work requiring more test data. Selecting the sandbox type therefore affects the amount of data that can normally be stored and retained in that environment. Salesforce's sandbox overview and comparison of sandbox types is available at Sandbox Types and Templates.

QUESTION NO: 7

A developer created a trigger on a custom object. This custom object also has some dependent pick lists.

According to the order of execution rules, which step happens first?

A. The original record is loaded from the database.

B. System validation is run for maximum field lengths.

C. Old values are overwritten with the new record values.

D. JavaScript validation is run in the browser.

ANSWER: D

Explanation:

JavaScript validation is run in the browser is correct because dependent picklists have a specific client-side validation step in Salesforce's save order of execution. Before the save request reaches Salesforce's servers and before server-side processing such as loading the existing database record, applying incoming field values, system validation, and trigger execution begins, the browser validates the dependency relationship between the controlling and dependent picklist values. This ensures that a value selected for a dependent picklist is valid for the current value of its controlling field. The behavior applies to saves initiated from the Salesforce user interface, where the browser can run the JavaScript validation. Server-originated operations, such as API requests and Apex DML, do not use a browser and therefore do not perform this client-side step; Salesforce instead applies its applicable server-side validation during processing. Because the question explicitly states that the custom object has dependent picklists, it signals this special pre-server validation event. Salesforce documents this ordering in its Order of Execution reference and provides additional details about field dependencies.

QUESTION NO: 8

A developer creates a Lightning web component that imports a method within an Apex class. When a Validate button is pressed, the method runs to execute complex validations.

In this implementation scenario, which two options are.. of the Controller according to the MVC architecture?

Choose 2 answers

A. HTML file

B. Apex class

C. JavaScript file

D. XML file

ANSWER: B C

Explanation:

The Apex class and JavaScript file perform the controller responsibilities in this implementation. The JavaScript file contains the component's client-side behavior: it responds to the Validate button interaction, invokes the imported Apex method, manages asynchronous results, and updates reactive component state that the template can render. Salesforce documents that a Lightning web component can import and call an Apex method from its JavaScript module, including calls initiated by user actions.

The Apex class provides the server-side behavior for validations that require Salesforce data access or more substantial business logic. To be callable by a Lightning web component, the method is exposed with `@AuraEnabled` and, when appropriate for read-only operations, `cacheable=true`. The Apex method receives the request from the component, executes the validation logic, and returns a result for the JavaScript code to process. Together, these client- and server-side layers coordinate user input, application behavior, and returned data, which corresponds to controller-style responsibilities in the MVC framing used by the question.

See Salesforce's documentation on calling Apex methods from Lightning web components and JavaScript in Lightning web components.

QUESTION NO: 9

Which two statements are true about Getter and Setter methods as they relate to Visualforce?

A. Setter methods always have to be declared global.

B. There is no guarantee for the order in which Getter methods are called.

C. A corresponding Setter method is required for each Getter method.

D. Getter methods pass values from a controller to a page.

ANSWER: B D

Explanation:

There is no guarantee for the order in which Getter methods are called because Visualforce evaluates expressions as it renders a page, and a getter can be invoked repeatedly whenever the framework needs the value. A controller therefore must not rely on one getter being called before another, or on a getter being called only once. Getter logic should be idempotent where possible and should avoid unintended state changes, SOQL queries, or DML operations that could produce inconsistent results or consume governor limits during repeated evaluation.

Getter methods pass values from a controller to a page because Visualforce uses getters to retrieve the value of a controller property referenced in page markup, such as a merge expression or component attribute. For example, a page reference to `{!accountName}` causes Visualforce to use the associated `getAccountName()` method (or an Apex property getter) to obtain the displayed value. This controller-to-page flow is the core purpose of getter methods in Visualforce. Salesforce documents the getter evaluation behavior and advises developers not to depend on invocation order: Visualforce Getter Methods. Related controller property conventions are described in the Apex Developer Guide.

QUESTION NO: 10

Management asked for opportunities to be automatically created for accounts with annual revenue greater than $1, 000,000. A developer created the following trigger on the Account object to satisfy this requirement.

```
for (Account a : Trigger.new) {

if (a.AnnualRevenue > 1000000) {

List < Opportunity > oppList = [SELECT Id FROM Opportunity WHERE AccountId = :a.Id];

if (oppList.size() == 0) {

Opportunity oppty = new Opportunity(Name = a.Name, StageName = ' Prospecting ' , CloseDate =
System.today().addDays(30));

insert oppty;

}

}

}
```

Users are able to update the account records via the UI and can see an opportunity created for high annual revenue accounts. However, when the administrator tries to upload a list of 179 accounts using Data Loader, it fails with system, Exception errors.

Which two actions should the developer take to fix the code segment shown above?

Choose 2. answers

A. Query for existing opportunities outside the for loop.

B. Check if all the required fields for Opportunity are being added on creation.

C. Move the DML that saves opportunities outside the for loop.

D. Use Database query to query the opportunities.

ANSWER: A C

Explanation:

Query for existing opportunities outside the for loop and Move the DML that saves opportunities outside the for loop are the required bulkification changes. Apex triggers can receive a collection of records in a single transaction, including records submitted through Data Loader. The trigger must therefore first collect the relevant Account IDs, perform one SOQL query that retrieves existing opportunities for all of those IDs, and use the query results to determine which accounts need a new opportunity. This prevents a separate query from being issued for every Account.

The code should also add each new Opportunity to a list and issue one `insert` operation after processing all records. This approach keeps the transaction within Salesforce governor limits, which limit synchronous Apex transactions to 100 SOQL queries and 150 DML statements. Processing 179 accounts with the shown implementation can exceed both limits, producing the observed system exceptions. Bulk SOQL and bulk DML are fundamental trigger design practices because they make the same logic work consistently for a single UI update and for large API or Data Loader batches. Salesforce documents these practices in its Apex trigger best practices and lists the applicable limits in its Apex governor limits reference.

QUESTION NO: 11

How should a custom user interface be provided when a user edits an Account in Lightning Experience?

A. Override the Account's Edit button with Lightning Flow

B. Override the Account's Edit button with Lightning Action

C. Override the Account's Edit button with Lightning page.

D. Override the Account's Edit button with Lightning component.

ANSWER: D

Explanation:

Override the Account's Edit button with Lightning component. In Lightning Experience, a standard object button such as Edit can be overridden to launch a custom Aura component, allowing the developer to replace the standard edit experience with a tailored user interface. The component is selected from the object's Buttons, Links, and Actions setup area as the Lightning Experience override for the standard Edit button. To be eligible for this use, the Aura component implements `lightning:actionOverride`. This interface identifies the component as one that can replace a standard action, including Edit, New, View, or Tab, in Lightning Experience and the Salesforce mobile app where supported. The component can then control the editing workflow, present custom fields or guidance, perform validation, and use Lightning Data Service or Apex as appropriate to save changes. Salesforce documents standard button overrides and their Lightning component requirements in its developer guidance. See lightning:actionOverride and Custom Action Overrides.

QUESTION NO: 12

Which three data types can a SOQL query return?

Choose 3 answers

A. Double

B. O Long

C. sObject

D. Integer

E. List

ANSWER: C D E

Explanation:

sObject, **List**, and **Integer** are valid Apex result types for SOQL queries, depending on the query form and intended result. A query expected to find one record can be assigned directly to a specific sObject type, such as `Account`, which is an sObject. For example, an Apex expression that queries a single Account record can assign that result to an `Account` variable. Most record queries are assigned to a collection, such as `List<Account>`; `List` is therefore the applicable collection data type represented by the choices.

A SOQL query using the `COUNT()` form can be assigned directly to an `Integer`. This supports efficient record counting without retrieving individual records, for example: `Integer contactCount = [SELECT COUNT() FROM Contact];`. Salesforce documents both standard SOQL query assignment patterns and the special count-query result behavior in its Apex SOQL guidance. See Salesforce Trailhead: Write SOQL Queries and Salesforce Apex Reference: SOQL Aggregate Functions.

QUESTION NO: 13

A developer needs to save a List of existing Account records named myAccounts to the database, but the

records do not contain Salesforce Id values. Only the value of a custom text field configured as an External ID with an API name of Foreign_Key c is known.

Which two statements enable the developer to save the records to the database without an Id? (Choose two.)

A. upsert myAccounts Account.Fields.Foreign_Key__c;

B. upsert myAccounts(Foreign_Key__c);

C. Database.upsert(myAccounts, Account.Fields.Foreign_Key__c);

D. Database.upsert(myAccounts).Foreign_Key__c;

ANSWER: A C

Explanation:

Both `upsert myAccounts Account.Fields.Foreign_Key__c;` and `Database.upsert(myAccounts, Account.Fields.Foreign_Key__c);` save the Account records by using the specified external ID field as the match key. An upsert operation first attempts to locate an existing Account whose `Foreign_Key__c` value matches the value on each record in `myAccounts`. When a match is found, Salesforce updates that Account; when no match exists, Salesforce inserts a new Account. Therefore, the records do not need Salesforce record IDs for this operation. The custom field must be configured as an External ID (and its values should uniquely identify the source-system records for predictable matching). The DML-statement form uses the `upsert` keyword directly, while the Database class form invokes the equivalent operation programmatically and can return `Database.UpsertResult` values for individual records. In both forms, `Account.Fields.Foreign_Key__c` supplies the external-ID field token required to identify the field used for matching.

Salesforce documents external-ID upsert behavior and the supported Apex DML syntax in its Apex Upsert examples and Database class reference.

QUESTION NO: 14

An Opportunity needs to have an amount rolled up from a custom object that is not in a master-detail relationship.

How can this be achieved?

A. Write a trigger on the child object and use a red-black tree sorting to sum the amount for all related child objects under the Opportunity.

B. Write a Process Builder that links the custom object to the Opportunity.

C. Write a trigger on the child object and use an aggregate function to sum the amount for all related child objects under the Opportunity

D. Use the Streaming API to create real-time roll-up summaries.

ANSWER: C

Explanation:

Writing a trigger on the child object and using an aggregate function to sum the amount for all related child objects under the Opportunity is the appropriate approach when the relationship is a lookup rather than master-detail. Native roll-up summary fields require a master-detail relationship, so a custom Apex implementation is needed to maintain the Opportunity's total.

A bulkified child-object trigger can collect the affected Opportunity IDs, including both old and new lookup values when the relationship or amount changes. It can then execute aggregate SOQL, such as `SELECT Opportunity__c, SUM(Amount__c) FROM Custom_Object__c WHERE Opportunity__c IN :opportunityIds GROUP BY Opportunity__c`, and update the corresponding Opportunity records with the calculated totals. The trigger should account for insert, update, delete, and undelete operations so the rollup remains accurate throughout the child record lifecycle. Aggregate queries are designed to calculate values such as `SUM()` efficiently across groups of records, while bulk trigger patterns ensure the implementation stays within Apex governor limits.

See Salesforce's documentation for SOQL aggregate functions and Apex trigger best practices.

QUESTION NO: 15

How does the Lightning Component framework help developers implement solutions faster?

A. By providing an Agile process with default steps

B. By providing code review standards and processes

C. By providing device-awareness for mobile and desktops

D. By providing change history and version control

ANSWER: C

Explanation:

By providing device-awareness for mobile and desktops is correct because the Lightning Component framework is designed to support responsive, device-aware user interfaces. Developers can build components that adapt their presentation and behavior to the form factor on which they run, including desktop and mobile experiences. This reduces the need to create and maintain separate implementations for different devices, allowing teams to reuse the same component-based solution across Salesforce user experiences.

Lightning components also use a client-side architecture and reusable component model, which lets developers assemble applications from encapsulated UI building blocks. Device awareness is especially valuable when a component must

account for differences in available screen space, interaction patterns, and the context in which Salesforce is being used. Salesforce provides form-factor capabilities so components can identify whether they are running in a large, medium, or small context and render an appropriate experience. This enables developers to deliver a consistent solution more quickly while still optimizing the interface for mobile and desktop users. See Salesforce's Lightning Web Components CSS and responsive design guidance and the form factor documentation.

QUESTION NO: 16

Which declarative process automation feature supports iterating over multiple records?

A. Flows

B. Validation Rules

C. Approval Process

D. Workflow rules

ANSWER: A

Explanation:

Flows support iterating over multiple records through the Loop element in Flow Builder. A flow can first retrieve a set of records with Get Records, store those results in a record collection variable, and then use a Loop element to process one item from that collection at a time. Within each iteration, the flow can evaluate values, assign changes to a separate collection, create related records, or perform other declarative actions appropriate to the business process. This capability is important when automation must apply consistent logic to a group of records rather than only the record that initiated the transaction. Salesforce recommends designing loops carefully: make required changes in assignments during the loop, add changed records to a collection, and perform bulk record updates after the loop where possible. This pattern helps keep the automation efficient and aligned with Salesforce's bulk-processing model. Flow Builder is Salesforce's primary declarative automation tool and provides purpose-built elements for collection processing, including Loop and Assignment. See Salesforce's Loop element reference and Trailhead guidance for looping over records.

QUESTION NO: 17

What are two benefits of using External IDs?

Choose 2 answers

A. An External ID field can be used to reference an ID from another external system.

B. An External ID can be a formula field ta help create a unique key from two fields in Salesforce.

C. An External ID can be used with Salesforce Mobile to make external data visible.

D. An External 1D is indexed and can improve the performance of SOOL queries.

ANSWER: A D

Explanation:

An External ID field can be used to reference an ID from another external system because it stores an identifier that an outside application already uses for the same record. This creates a reliable matching key between systems, allowing integrations and data-loading tools to locate existing Salesforce records rather than creating duplicates. For example, an integration can use an account number or customer identifier maintained by an ERP system to perform an upsert, which updates a matching Salesforce record or inserts a record when no match exists.

An External 1D is indexed and can improve the performance of SOOL queries because Salesforce automatically indexes fields designated as External IDs. An index helps Salesforce selectively locate matching rows instead of scanning a much larger set of records, which is especially valuable for integration queries and lookups that filter by a source-system identifier. External ID fields therefore support both dependable record matching and efficient access to records at scale. Salesforce

documents External IDs as fields used to contain unique record identifiers from systems outside Salesforce and notes their indexed status. See Salesforce Field Types Reference and Salesforce Upsert API Reference.

QUESTION NO: 18

What is a key difference between a Master-Detail Relationship and a Lookup Relationship?

A. A Master-Detail Relationship detail record inherits the sharing and security of its master record.

B. When a record of a master object in a Lookup Relationship is deleted, the detail records are also deleted.

C. A Lookup Relationship is a required field on an object.

D. When a record of a master object in a Master-Detail Relationship is deleted, the detail records are kept and not deleted.

ANSWER: A

Explanation:

A Master-Detail Relationship detail record inherits the sharing and security of its master record. In Salesforce, a master-detail relationship creates a strong parent-child dependency: the detail record does not have an independent owner and its access is controlled by the master record's ownership and sharing settings. This relationship behavior helps ensure that users who can access the master record receive the appropriate corresponding access to its related detail records. It also supports the tightly coupled data model used for related records that should be managed as part of a parent record.

This inheritance is a defining characteristic of master-detail relationships and is important when designing an application's record-access model. Salesforce documentation describes master-detail relationships as relationships in which the detail record inherits sharing and security settings from the master record. Developers should account for this behavior when determining whether related records need independent ownership or independently configurable access. See Salesforce Help: Object Relationships Overview and Salesforce Developer Guide: Relationships.

QUESTION NO: 19

which statement is true regarding execution order when triggers are associated to the same object and event?

A. Trigger execution order cannot be guaranteed.

B. executed In the order they are modified.

C. Triggers are executed alphabetically by trigger name.

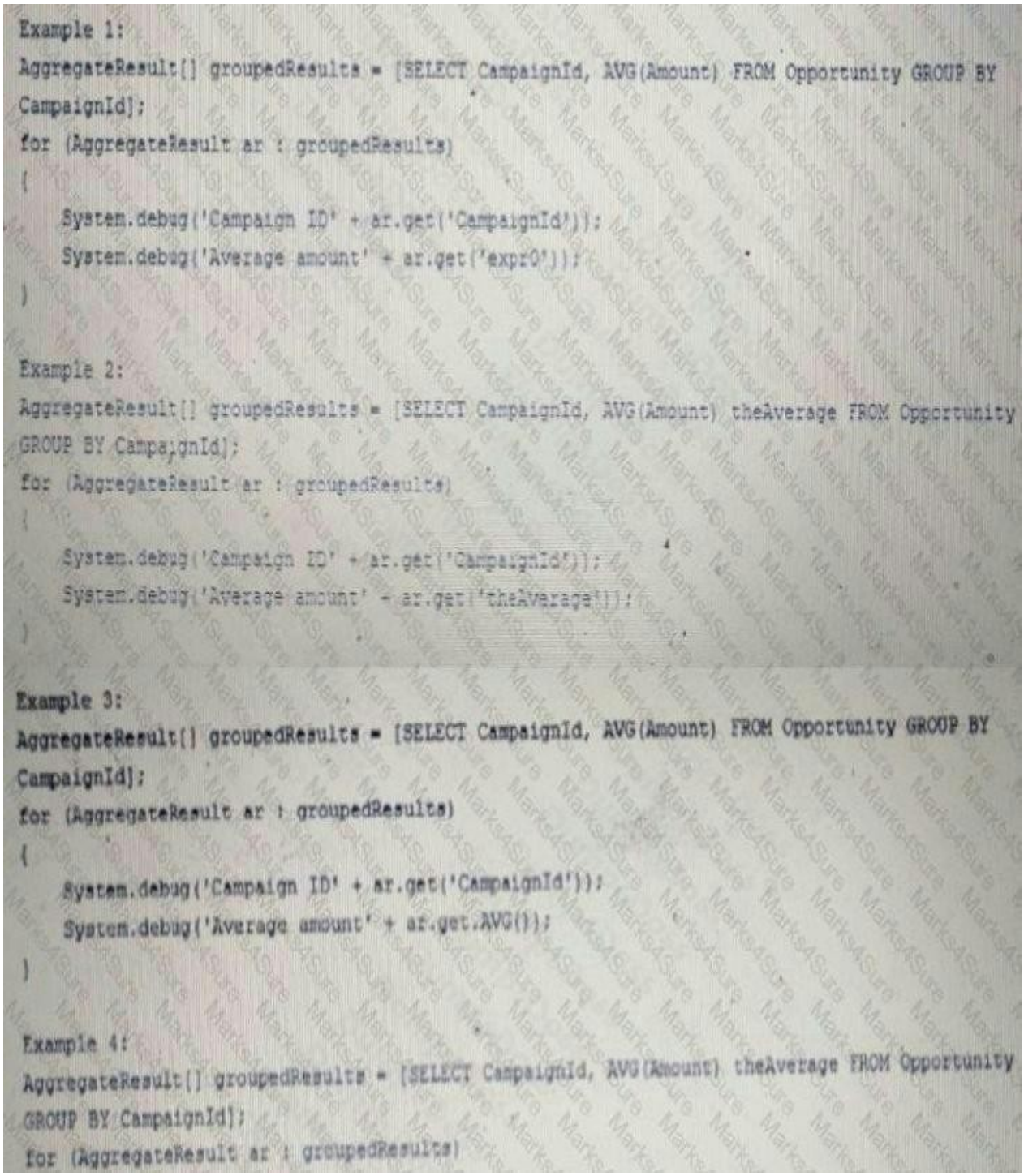
D. Triggers are executed in the order they are created.

ANSWER: A

Explanation:

Trigger execution order cannot be guaranteed. When more than one Apex trigger is defined for the same object and the same event, such as multiple before-insert or after-update triggers, Salesforce does not document or promise a consistent sequence in which those separate triggers run. A developer must therefore treat their relative execution order as nondeterministic and must not build dependencies in which one trigger assumes another trigger has already set a field, created related data, or performed a validation-related action. Salesforce's Apex Developer Guide explicitly states that the order of execution for multiple triggers on the same object and event is not guaranteed. The recommended design is to use a single trigger per object and delegate logic to handler classes, allowing the application to define and maintain an intentional order among its own units of work. This approach also makes bulk processing, testing, recursion control, and long-term maintenance more reliable. See the Salesforce guidance on Apex trigger order of execution and its recommendation for Apex trigger best practices.

QUESTION NO: 20

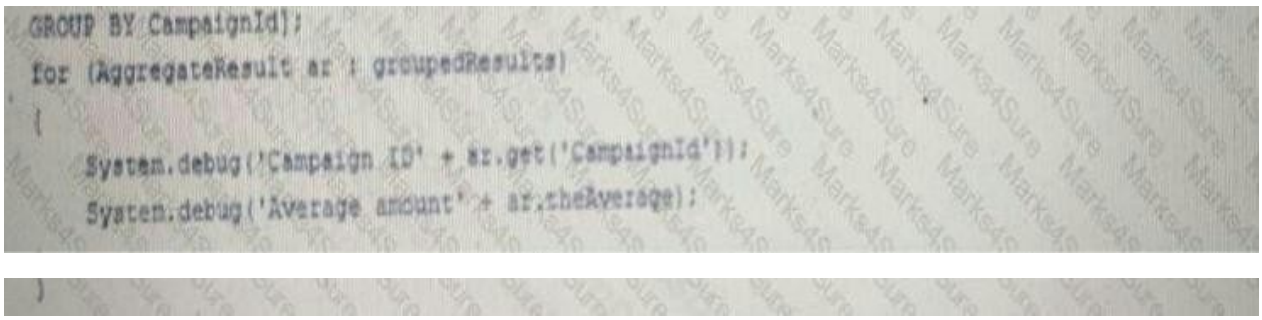


```
Example 1:
AggregateResult[] groupedResults = [SELECT CampaignId, AVG(Amount) FROM Opportunity GROUP BY
CampaignId];
for (AggregateResult ar : groupedResults)
{
    System.debug('Campaign ID' + ar.get('CampaignId'));
    System.debug('Average amount' + ar.get('expr0'));
}

Example 2:
AggregateResult[] groupedResults = [SELECT CampaignId, AVG(Amount) theAverage FROM Opportunity
GROUP BY CampaignId];
for (AggregateResult ar : groupedResults)
{
    System.debug('Campaign ID' + ar.get('CampaignId'));
    System.debug('Average amount' + ar.get('theAverage'));
}

Example 3:
AggregateResult[] groupedResults = [SELECT CampaignId, AVG(Amount) FROM Opportunity GROUP BY
CampaignId];
for (AggregateResult ar : groupedResults)
{
    System.debug('Campaign ID' + ar.get('CampaignId'));
    System.debug('Average amount' + ar.get.AVG());
}

Example 4:
AggregateResult[] groupedResults = [SELECT CampaignId, AVG(Amount) theAverage FROM Opportunity
GROUP BY CampaignId];
for (AggregateResult ar : groupedResults)
```



```
GROUP BY CampaignId];
for (AggregateResult ar : groupedResults)
{
    System.debug('Campaign ID' + ar.get('CampaignId'));
    System.debug('Average amount' + ar.theAverage);
}
```

Which two examples above use the system. debug statements to correctly display the results from the SOQL aggregate queries? Choose 2 answers

A. Example 1

B. Example 2

C. Example 3

D. Example 4

ANSWER: B C

Explanation:

Example 2 and Example 3 correctly use `System.debug` with aggregate-query results. SOQL aggregate functions, such as `COUNT()`, `SUM()`, `AVG()`, `MIN()`, and `MAX()`, return aggregate values rather than ordinary sObject field values. When an aggregate query includes grouping or aliases, Apex represents each returned row as an `AggregateResult`. A debug statement can display the aggregate result collection directly, or it can display an individual aggregate value retrieved from an `AggregateResult` by its alias (or by the generated expression name when no alias was supplied). These approaches let a developer inspect the values calculated by the query in the execution log. Example 2 uses a debug statement in a form appropriate for the aggregate result returned by the query, and Example 3 likewise outputs the relevant aggregate-query result correctly. This is useful when validating grouped totals, record counts, and other summary calculations during Apex development. Salesforce documents aggregate-query return values and the use of aliases in SOQL Aggregate Functions. The platform's logging behavior and `System.debug` usage are documented in the System Class reference.

QUESTION NO: 21

Which code statement includes an Apex method named updateAccounts in the class AccountController for use in a Lightning web component?

A. import updateAccounts from " AccountControlles ' ;

B. import updateAccounts from " Salesforce/apex/AccountController: ' ;

C. import updateAccounts from " Account@ontroller.updateAccounts ' ;

D. import updateAccounts from " @salesforce/apex/AccountController.updateAccounts " ;

ANSWER: D

Explanation:

The statement `import updateAccounts from '@salesforce/apex/AccountController.updateAccounts';` is the correct way to make an Apex method available to a Lightning web component JavaScript module. Lightning Web Components use the scoped `@salesforce/apex` module identifier to import Apex. The portion following it identifies the Apex class and method in the required `ClassName.methodName` format. In this case, `AccountController` is the Apex class and `updateAccounts` is the method being imported; the local JavaScript identifier is also named `updateAccounts`, allowing the component to invoke it imperatively or pass it to a wire adapter when its signature and use case support that approach.

For an Apex method to be callable by an LWC, it must be declared `static`, `public` or `global`, and annotated with `@AuraEnabled`. If the component needs to call the method from JavaScript, the import itself does not execute Apex; it simply provides a reference that the component can call, typically returning a Promise for imperative invocations. Salesforce documents this Apex import pattern and the required server-side method exposure in its Lightning Web Components developer guidance: Import Apex Methods into Lightning Web Components and Call Apex Methods.

QUESTION NO: 22

Universal Container use a simple order Management app. On the Order Lines, the order line total is calculated by multiplying the item price with the quantity ordered. There is a Master-Detail relationship between the Order and the Order Lines object.

What is the practice to get the sum of all order line totals on the order header?

A. Roll-Up Summary Field

B. Apex Trigger

C. Process Builder

D. Declarative Roll-Up Summaries App

ANSWER: A

Explanation:

Roll-Up Summary Field is the appropriate declarative feature for displaying the total of all Order Line totals on the related Order record. A roll-up summary field is created on the master object, which in this case is Order, and can calculate aggregate values from its related detail records, the Order Lines. Configuring the field with the *SUM* calculation type and selecting the Order Line total currency or number field causes Salesforce to maintain the Order header value automatically as Order Lines are created, edited, deleted, or undeleted.

This is specifically supported because the relationship is master-detail. Roll-up summary fields are designed to summarize detail-record data on the master record, including counts, sums, minimum values, and maximum values. The calculation is maintained by the platform, so the Order total remains current without custom code or a separate automation process. This approach is also easier to administer and follows the platform preference of using declarative functionality when it directly satisfies the requirement. Salesforce documents the available summary operations and the requirement to create roll-up summary fields on the master side of a master-detail relationship in its Roll-Up Summary Fields documentation and the Trailhead roll-up summary fields unit.

QUESTION NO: 23

Assuming that name is a String obtained by an tag on a Visualforce page, which two SOQL queries performed are safe from SOQL injection? Choose 2 answers

A)

B)

C)

D)

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: B C

Explanation:

The safe approaches are the query that applies `String.escapeSingleQuotes()` to the value received from the Visualforce page and the query that uses a SOQL bind variable. Escaping single quotes ensures that quote characters supplied in a string value are treated as literal data rather than allowing the value to terminate a quoted SOQL string literal and alter the query structure. This is particularly important when a dynamic SOQL statement is assembled as a string.

A bind variable is the preferred approach whenever it can be used. In static SOQL, the `:name` syntax passes the value separately from the query syntax, so Salesforce handles the value as data rather than executable SOQL text. This eliminates the need to concatenate untrusted input into the statement and provides protection even when the input contains characters with special meaning in SOQL. Salesforce's secure-coding guidance identifies bind variables as a primary defense and documents `escapeSingleQuotes` as an appropriate mitigation when dynamic SOQL construction is necessary. See Salesforce SOQL and SOSL Injection Prevention and Salesforce String.escapeSingleQuotes documentation.

QUESTION NO: 24

When using SalesforceDX, what does a developer need to enable to create and manage scratch orgs?

A. Production

B. Dev Hub

C. Environment Hub

D. Sandbox

ANSWER: B

Explanation:

A developer must enable Dev Hub to create and manage Salesforce DX scratch orgs. Dev Hub is a capability enabled in a production org or a business org that serves as the central control point for the scratch-org lifecycle. It authorizes Salesforce CLI and other supported tooling to create scratch orgs, tracks the scratch orgs associated with the hub, and provides access to Dev Hub features such as package and namespace management where applicable. After Dev Hub is enabled, the developer authenticates to that org and designates it as the Dev Hub when running Salesforce CLI commands. Scratch org definitions then specify the edition, features, settings, and configuration required for each temporary development environment. Scratch orgs are intended to be source-driven and disposable, allowing teams to create consistent environments for development, testing, and continuous integration. Enabling Dev Hub is therefore the required organizational setup before a developer can use Salesforce DX to provision and administer scratch orgs. Salesforce documents this requirement in its scratch-org overview and setup guidance: Scratch Orgs and Set Up Dev Hub.

QUESTION NO: 25

What are two ways for a developer to execute tests in an org?

Choose 2 answers

A. Tooling API

B. Metadata API

C. Bulk API

D. Developer Console

ANSWER: A D

Explanation:

Tooling API and **Developer Console** are supported ways to execute Apex tests in a Salesforce org. The Tooling API exposes test-execution resources programmatically, including asynchronous test runs through the `runTestsAsynchronous` endpoint. This enables a developer or an external development tool to submit selected test classes or methods and then retrieve status and result information. It is particularly useful for automated development workflows and integrations that need to trigger tests without manually using the Salesforce user interface.

The Developer Console provides an interactive, browser-based environment within Salesforce for developers to run Apex tests. From its Test menu, a developer can run all tests, execute selected tests, view results, and inspect associated debug logs. This supports rapid validation of Apex code directly in the org during development and troubleshooting. Salesforce documents test execution through Developer Console and other development tools, while the Tooling API documentation describes the programmatic test-running capability. See Salesforce Apex Testing Tools and Tooling API AsyncApexJob reference.

QUESTION NO: 26

Which three Salesforce resources can be accessed from a Lightning web component?

Choose 3 answers

A. SVG resources

B. Third-party web components

C. Content asset files

D. Static resources

E. All external libraries

ANSWER: A C D

Explanation:

Lightning web components can access **SVG resources**, **Content asset files**, and **Static resources** through Salesforce-supported resource mechanisms. Static resources are uploaded to Salesforce and are exposed to a component by importing the resource URL from the `@salesforce/resourceUrl` scoped module. This is the standard approach for component assets such as images, stylesheets, JavaScript libraries, and packaged files. SVG files can likewise be delivered as component assets, allowing a component to use scalable graphic content while keeping the asset managed in Salesforce. Content assets are files stored and managed in Salesforce Content; a component can obtain their URLs with the `@salesforce/contentAssetUrl` scoped module. This is especially useful when reusable images or other approved content files are maintained as Salesforce content assets rather than as static resources. These supported import mechanisms let the component reference platform-managed files using URLs that Salesforce resolves at runtime, including the appropriate namespace and versioning details. See Salesforce's documentation for accessing static resources and accessing content asset files.

QUESTION NO: 27

A company ' s engineering department is conducting a month-long test on the scalability of an in-house-developed software that requires a cluster of 100 or more servers. Which of the following models is the best to use?

A. PaaS

B. SaaS

C. BaaS

D. IaaS

ANSWER: D

Explanation:

IaaS is the best fit because the engineering department needs temporary access to a large, configurable pool of server capacity for an in-house application's month-long scalability test. Infrastructure as a Service supplies fundamental computing resources—such as processing, storage, and networking—on demand. The team can provision a cluster with 100 or more virtual servers, select suitable machine sizes and network configurations, install its own operating systems and test software, and scale capacity up or down as test scenarios require. This avoids the capital cost, procurement delay, data-center space, and ongoing maintenance involved in acquiring enough physical hardware for a short-lived test. When the testing period ends, the provisioned resources can be decommissioned, so billing generally corresponds to the period and capacity actually used. This model also preserves the infrastructure-level control that is valuable when measuring the behavior of custom software under high concurrency, load, or distributed-system conditions. The National Institute of Standards and Technology defines IaaS as consumer provisioning of processing, storage, networks, and other fundamental computing resources, with control over deployed software and selected networking components. See the NIST cloud-computing definition and AWS's IaaS overview.

QUESTION NO: 28

A company has a custom object, order __c, that has a required, unique external ID field called order Number__c.

Which statement should be used to perform the DML necessary to insert new records and update existing records in a list of Order __c records

using the external ID field?

A. merge orders;

B. merge orders Order Number_c;

C. upsert orders Order_Number__c;

D. upsert orders;

ANSWER: C

Explanation:

`upsert orders Order_Number__c;` is the appropriate Apex DML statement because it explicitly directs Salesforce to use the `Order_Number__c` external ID field as the matching key for the records in `orders`. An upsert operation combines insert and update behavior in one statement: Salesforce updates a database record when a submitted record's external ID value matches an existing record, and inserts the submitted record when no matching external ID value exists. This is particularly useful for integrations and data synchronization processes, where an external business identifier, such as an order number, is available even when the Salesforce record ID is not. The external ID field is required and unique, so each supplied value can identify at most one existing Order record, making it suitable for this operation. Specifying the field in the statement is important because the requirement is to match using the external ID rather than the Salesforce record `Id`. Salesforce documents the Apex upsert syntax as `upsert sObjectList externalIdField` when an external ID is used for matching. See the Salesforce Apex upsert documentation and the Salesforce field type reference for external IDs.

QUESTION NO: 29

Which two events need to happen when deploying to a production org? Choose 2 answers

A. All triggers must have at least 1% test coverage.

B. Apex code in the org must have at least 75% test coverage overall.

C. All triggers must have at least 75% test coverage.

D. All test and triggers must have at least 75% test coverage combined

ANSWER: A B

Explanation:

A production deployment that includes Apex requires Salesforce to run tests and enforce its code-coverage deployment requirements. Every Apex trigger must receive some test coverage—at least 1%—so that no trigger is entirely untested. This is a distinct requirement from the overall coverage threshold and applies to each trigger individually.

In addition, the org's Apex code must have at least 75% aggregate test coverage after the deployment. Salesforce calculates this as an organization-wide coverage percentage, rather than requiring every individual class or trigger to independently meet 75%. The deployment is permitted only when the tests run successfully and the resulting aggregate coverage satisfies that minimum. These rules help ensure that production Apex changes have meaningful automated validation while allowing coverage to be distributed across the codebase. Salesforce documents both the 75% organization-wide threshold and the minimum coverage requirement for each trigger in its deployment guidance. See Deploying Apex and Testing Apex.

QUESTION NO: 30

Where are two locations a developer can look to find information about the status of batch or future methods?

Choose 2 answers

A. Developer Console

B. Apex Jobs

C. Paused Flow Interviews component

D. Apex Flex Queue

ANSWER: A B

Explanation:

Developer Console and **Apex Jobs** are the appropriate places to monitor asynchronous Apex processing. In the Developer Console, the Jobs tab provides visibility into asynchronous Apex work, including submitted Batch Apex jobs and future-method invocations. A developer can use this interface while developing or troubleshooting to review job information and correlate work with execution details available through logs when logging is enabled.

The Apex Jobs page in Setup is the central administrative monitoring page for asynchronous Apex. It exposes job records and their processing state, such as whether work is queued, processing, completed, or failed, and provides operational details including submission and completion information. This makes it useful for confirming whether Batch Apex or future work ran and for investigating failures in a production-oriented administration workflow.

Salesforce documents asynchronous Apex monitoring through the Apex Jobs page and identifies the Developer Console Jobs tab as another way to view asynchronous Apex jobs. See Monitor Asynchronous Apex and Debug Apex.

QUESTION NO: 31 - (SIMULATION)

what are the methods used to show input in classic and lightning ?

ANSWER: See the explanation for the answer

Explanation:

Classic (Visualforce): Use Visualforce input components, such as `<apex:inputText>`, `<apex:inputField>`, `<apex:inputCheckbox>`, and `<apex:inputTextarea>`.

```
<apex:inputField value="{!Account.Name}" />
```

Lightning: Use Lightning base input components. In Aura, use `<lightning:input>` (or field components such as `<lightning:inputField>` within `<lightning:recordEditForm>`). In LWC, use `<lightning-input>` or `<lightning-input-field>`.

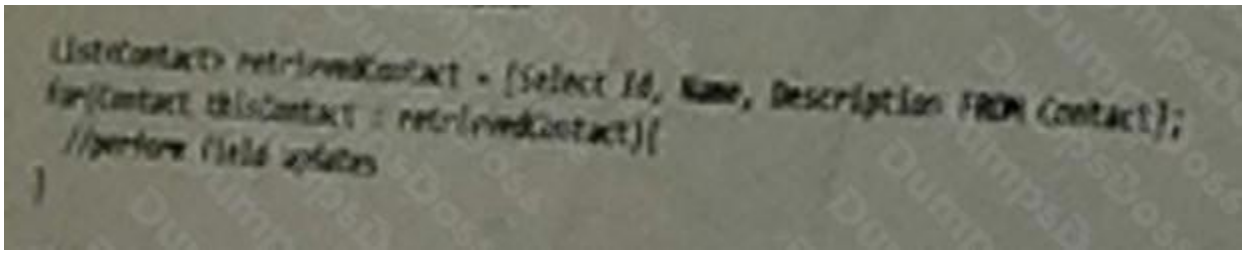
```
<!-- LWC -->
<lightning-input label="Name" value={name} onchange={handleChange}></lightning-input>

<!-- LWC record form -->
<lightning-input-field field-name="Name"></lightning-input-field>
```

Visualforce Classic: `apex:inputText` / `apex:inputField`.
Lightning Aura: `lightning:input` / `lightning:inputField`.
Lightning Web Components: `lightning-input` / `lightning-input-field`.

QUESTION NO: 32

As a part of class implementation a developer must execute a SOQL query against a large data ser based on the contact object. The method implementation is as follows.



Which two methods are best practice to implement heap size control for the above code? (Choose 2 Answers)

A. Use the FOR UPDATE option on the SOQL query to lock down the records retrieved.

B. Use transient keyword when declaring the retrieve variable.

C. Use a SOQL FOR loop, to chunk the result set in batches of 200 records.

D. Use WHERE clauses on the SOQL query to reduce the number of records retrieved.

ANSWER: C D

Explanation:

Using a SOQL `for` loop is a heap-efficient pattern for processing a large Contact query result. Rather than assigning every queried record to a collection at once, Apex retrieves and processes records in internally managed batches of 200 records. This substantially limits the amount of queried data that must be held in application heap memory at a given time and is the standard approach when the code needs to iterate through many rows.

Using `WHERE` clauses to reduce the number of records retrieved is also a direct heap-size control. Heap consumption includes the records and field values materialized by the query, so selecting only the Contacts that the method actually needs reduces memory use before processing starts. A selective filter can additionally improve query efficiency and reduce the overall work performed by the transaction. These approaches can be used together: filter the query to the required Contact population, then process that population through a SOQL `for` loop. Salesforce documents that SOQL `for` loops retrieve records in batches of 200 and identifies heap size as an Apex governor limit. See SOQL For Loops and Apex Governor Limits.

QUESTION NO: 33

Which scenario is valid for execution by unit tests?

A. Load data from a remote site with a callout.

5. Set the created date of a record using a system method.

Cc: Execute anonymous Apex as a different user.

B. Generate a Visualforce PDF with geccontentAsPDF ().

C. Set the created date of a record inserted in the test by using Test.setCreatedDate.

ANSWER: C

Explanation:

Setting the created date of a record through `Test.setCreatedDate(recordId, createdDatetime)` is valid in an Apex unit test. This Test class method is specifically designed to let a test control the `CreatedDate` value of a record that was inserted during that test. The record must first be created and inserted, and its Id is then passed to `Test.setCreatedDate` along with the desired `Datetime`. This is useful when application logic depends on record age, date-based calculations, aging thresholds, or queries and behavior involving `CreatedDate`. It lets the test establish deterministic time-based conditions without attempting to directly assign a value to the system-maintained `CreatedDate` field during DML. The method is available only in test context, which keeps this ability isolated from production code and preserves the normal platform behavior for audit fields outside tests. Salesforce documents `Test.setCreatedDate` as a

Test class method for setting the created date of a test record; see the Apex Test Class reference. For the broader conventions around creating reliable Apex tests and test data, see Testing Apex.

QUESTION NO: 34

The following automations already exist on the Account object;

• A workflow rule that updates a field when a certain criteria is met

• A custom validation on a field

• A How that updates related contact records

A developer created a trigger on the Account object.

What should the developer consider while testing the trigger code?

A. The flow may be launched multiple times.

B. Workflow rules will fire only after the trigger has committed all DML operations to the database.

C. A workflow rule field update will cause the custom validation to run again.

D. The trigger may fire multiple times during a transaction.

ANSWER: D

Explanation:

The trigger may fire multiple times during a transaction. Salesforce save processing can perform additional updates after the trigger's initial execution. In particular, when a workflow rule performs a field update on the same Account record, Salesforce runs the applicable before-update and after-update triggers one additional time as part of that save operation. Trigger code and its tests must therefore be designed with re-entry in mind: logic should remain correct if invoked again for the same record, avoid unintended duplicate work, and not rely on static assumptions that a trigger executes only once per transaction.

This consideration is especially important when the trigger itself performs record updates or interacts with declarative automation that can result in further saves. Tests should create data that satisfies the workflow criteria, execute the Account update, and assert the final persisted state after all synchronous automation has completed. The implementation should also use bulk-safe collection-based patterns so that repeated execution continues to respect governor limits and handles all records in the trigger context consistently. Salesforce documents that workflow field updates can cause before-update and after-update triggers to execute again; see the Order of Execution documentation and the Apex Trigger Best Practices guide.

QUESTION NO: 35

Assuming that 'name; is a String obtained by an tag on a Visualforce page.

Which two SOQL queries performed are safe from SOQL injections? Choose 2 answers

A. String query = 'SELECT Id FROM Account WHERE Name LIKE \"%' + name.noQuotes() + '%\"; List results = Database.query(query);

B. String query = 'SELECT Id FROM Account WHERE Name LIKE \"%' + String.escapeSingleQuotes(name) + '%\";

List results = Database.query(query);

C. String query = 'SELECT Id FROM Account WHERE Name LIKE \"%' + name + '%\"; List results = Database.query(query);

D. String query = '%' + name + '%';

List results = [SELECT Id FROM Account WHERE Name LIKE :query];

ANSWER: B D

Explanation:

`String.escapeSingleQuotes(name)` is safe in this dynamic SOQL construction because it escapes any single quotation marks supplied in the user-controlled string before that value is concatenated into a quoted SOQL string literal. As a result, quote characters in the input are treated as data within the `Name LIKE` value rather than allowing the input to terminate the literal and alter the query syntax. Salesforce specifically documents `escapeSingleQuotes` as an Apex method for escaping quote characters when dynamic SOQL must be constructed as a string.

Using a bind variable in static SOQL is also safe. In the query `[SELECT Id FROM Account WHERE Name LIKE :query]`, Apex supplies the value of `query` separately from the query's syntax. The database therefore interprets the entire user-derived value, including the surrounding percent characters used for the intended substring match, as a bind value rather than executable SOQL text. Bind variables are the preferred approach whenever the query structure is known at compile time because they avoid string concatenation for user input. Salesforce's SOQL injection guidance recommends bind variables and describes escaping quotes as an appropriate protection when dynamic SOQL is required. See Salesforce Secure Coding Guide: SOQL Injection and Dynamic SOQL in Apex.

QUESTION NO: 36

A Salesforce Administrator is creating a record-triggered flow. When certain criteria are met, the flow must call an Apex method to execute complex validation involving several types of objects.

When creating the Apex method, which annotation should a developer use to ensure the method

Can be used within the flow?

A. @future

B. @RemoteAction

C. @InvocableMethod

D. @AuraEnaled

ANSWER: C

Explanation:

`@InvocableMethod` is the Apex annotation that exposes a static method as an invocable action for declarative tools, including Flow Builder. After the Apex class and its invocable method are saved, the administrator can add an Apex action element to the record-triggered flow and select that method. This is designed for cases where Flow needs to delegate logic that is too complex or unsuitable for configuration-only flow elements, such as validation that evaluates related records across several object types.

An invocable method must be `static` and is called in bulk, receiving a list of request values and ordinarily returning a list of corresponding results when output is needed. For inputs and outputs beyond primitive types, developers can define an Apex request or response class and expose its fields with `@InvocableVariable`. Designing the action for collection-based processing is important because record-triggered flows can execute for multiple records in one transaction. Salesforce documents the requirements and supported use of invocable methods in the InvocableMethod Annotation reference and provides implementation guidance in its Apex Developer Guide documentation for invocable methods.

QUESTION NO: 37

Which three code lines are required to create a Lightning component on a Visualforce page? Choose 3 answers

A. $Lightning.createComponent

B.

C. $Lightning.use

ANSWER: A B C

Explanation:

Embedding a Lightning component in a Visualforce page requires the Visualforce page to load the Lightning Component framework, initialize the framework for the appropriate Lightning app, and then create the component. `<apex:includeLightning/>` is the Visualforce tag that makes the required Lightning JavaScript resources available on the page. The `$Lightning.use()` call initializes Lightning Out and specifies the dependency app that provides the component and its dependencies. Its callback runs only after the framework has initialized successfully. Within that callback, `$Lightning.createComponent()` instantiates the specified Aura component and places it into a target HTML element in the Visualforce page. Together, these lines establish the required load, initialization, and component-creation sequence. Salesforce documents this Lightning Out pattern with `<apex:includeLightning/>`, `$Lightning.use()`, and `$Lightning.createComponent()` in its Visualforce integration example. See Use Lightning Components in Visualforce Pages and Lightning Out.

QUESTION NO: 38

What are two ways a developer can get the status of an enqueued job for a class that implements the queueable interface?

Choose 2 answers

A. View the Apex Status page

B. View the Apex Jobs page

C. Query the AsyncApexJob object

D. View the Apex Flex Queue

ANSWER: B C

Explanation:

Viewing the Apex Jobs page and querying the AsyncApexJob object are both supported ways to obtain the status of work enqueued by a class that implements the Queueable interface. When Queueable Apex is submitted with `System.enqueueJob()`, Salesforce returns an asynchronous job ID. Salesforce creates a corresponding AsyncApexJob record, which holds operational information for the job, including its current `Status`, job type, creation time, completion time, and any exception details. A developer can query this object in SOQL, for example by filtering on the returned job ID, to programmatically monitor the job's progress or inspect its outcome. The Apex Jobs page in Setup presents these asynchronous Apex job records in the user interface, allowing an authorized user to review queued, processing, completed, failed, or aborted Queueable jobs without writing code. These approaches support both administrative monitoring and programmatic status checks for Queueable Apex. Salesforce documents Queueable Apex job monitoring and the asynchronous Apex job record model in the Queueable Apex documentation and the AsyncApexJob object reference.

QUESTION NO: 39

Universal Containers is developing a new Lightning web component for their marketing department. They want to ensure that the component is fine-tuned and provides a seamless user experience.

What are some benefits of using the Lightning Component framework?

A. Better performance due to client-side rendering

B. Automatic support for accessibility standards

C. Compatibility with all web browsers

D. Easy integration with third-party libraries

ANSWER: A B D

Explanation:

Better performance due to client-side rendering is a core benefit of the Lightning Component framework. Lightning web components use standard browser capabilities and render UI updates in the client, helping provide responsive interactions while minimizing unnecessary server round trips. This component model is designed to support modern, performant Salesforce user interfaces.

Automatic support for accessibility standards is also a framework benefit in the sense that Salesforce Lightning base components and the Lightning Design System incorporate accessible patterns, including appropriate ARIA support, keyboard interaction behavior, and accessible visual design. Developers can build on these provided patterns to deliver experiences that align with Salesforce accessibility guidance.

Easy integration with third-party libraries is supported through JavaScript and static resources. A Lightning web component can load a JavaScript or CSS library uploaded as a static resource by using `lightning/platformResourceLoader`. This enables teams to incorporate approved external functionality while retaining the component architecture and Salesforce security model. See Salesforce's Lightning Web Components documentation, the guidance for third-party JavaScript libraries, and the Lightning Design System accessibility resources.

QUESTION NO: 40

What are two considerations for deploying from a sandbox to production?

Choose 2 answers

A. At least 75% of Apex code must be covered by unit tests.

B. Unit tests must have calls to the System.assert method.

C. Should deploy during business hours to ensure feedback can be quickly addressed.

D. All triggers must have at least one line of test coverage.

ANSWER: A D

Explanation:

At least 75% of Apex code must be covered by unit tests. is a required deployment consideration because Salesforce runs tests during production deployments and requires the aggregate Apex test coverage to be at least 75%. Coverage is calculated across the organization's Apex code after the relevant tests run, so developers should validate deployment test results in the sandbox and ensure that the deployment will meet this production requirement.

All triggers must have at least one line of test coverage. is also required. In addition to the overall 75% Apex coverage threshold, Salesforce requires every Apex trigger to have some test coverage. This prevents a deployment from succeeding solely on high aggregate coverage when an individual trigger has never been exercised by a test. A sound deployment process therefore includes tests that invoke each trigger's applicable insert, update, delete, or undelete behavior, as well as assertions that verify the intended outcome. These requirements apply when deploying Apex and triggers to a production organization, including via change sets, Metadata API, Salesforce CLI, and other supported deployment mechanisms. Salesforce documents the production code-coverage rules in its Apex testing guidance: Apex Testing Best Practices and Testing Apex.

QUESTION NO: 41

Universal Containers (UC) processes orders in Salesforce in a custom object, Order__c. They also allow sales reps to upload CSV files with thousands of orders at a time.

A developer is tasked with integrating orders placed in Salesforce with UC ' s enterprise resource planning (ERP) system.

After the status for an Order__c is first set to ' Placed ' , the order information must be sent to a REST endpoint in the ERP system that can

process one order at a time.

What should the developer implement to accomplish this?

A. Callout from a Queueable class called from a trigger

B. Callout from a Batchable class called from a scheduled job

C. Flow with a callout from an invocable method

D. Callout from an @future method called from a trigger

ANSWER: A

Explanation:

Callout from a Queueable class called from a trigger is the appropriate design because the trigger can detect the first transition of an `Order__c` record's status to `Placed`, while the HTTP integration runs asynchronously outside the trigger transaction. The Queueable class implements `Database.AllowsCallouts`, which permits it to send the order payload to the ERP REST endpoint without making the save operation wait for the remote system's response. This protects the user-facing transaction and avoids synchronous-callout constraints in trigger execution.

The trigger should be bulkified: it evaluates all incoming records and identifies only records whose prior status was not `Placed`. The asynchronous work can then serialize processing so each ERP request contains one order. Where a larger imported set must be drained sequentially, a Queueable job can retain the remaining order identifiers and chain the next job after each successful one-order callout. Queueable Apex supports complex data types, monitoring through `AsyncApexJob`, and job chaining, making it well suited to reliable asynchronous integration patterns. Salesforce documents Queueable Apex and callout support at Queueable Apex and describes asynchronous Apex callouts at Asynchronous Callouts.

QUESTION NO: 42

Which two are best practices when it comes to Aura component and application event handling?

Choose 2 answers

A. Try to use application events as opposed to component events.

B. Reuse the event logic in a component bundle, by putting the logic in the helper.

C. Use component events to communicate actions that should be handled at the application level.

D. Handle low-level events in the event handler and re-fire them as higher-level events.

ANSWER: B D

Explanation:

"Reuse the event logic in a component bundle, by putting the logic in the helper" is a best practice because a helper centralizes JavaScript behavior that can be invoked by multiple controller actions or event handlers in the same Aura component bundle. This keeps event-handling code maintainable, avoids duplication, and lets the client-side controller remain focused on routing framework actions to reusable implementation logic. Salesforce describes helpers as a location for reusable client-side JavaScript functions.

"Handle low-level events in the event handler and re-fire them as higher-level events" is also a best practice because it provides useful abstraction between component internals and consumers. A component can interpret a detailed UI or child-component event locally, then expose a meaningful business-level event with a stable contract. This reduces coupling: parent components and application-level listeners respond to the intended action rather than needing knowledge of low-level implementation details. It also supports component encapsulation as the implementation evolves. Salesforce's Aura event guidance distinguishes component communication patterns and recommends designing events around the level of abstraction appropriate to their consumers. See the Salesforce event best-practices guide and the client-side controller and helper documentation.

QUESTION NO: 43

Which three data types can a SOQL query return? Choose 3 answers

A. List

B. Long

C. Integer

D. sObject

ANSWER: A C D

Explanation:

SOQL supports assignment to a `List` when a query can return multiple records. This is the usual form for a standard record query, such as `List<Account> accounts = [SELECT Id, Name FROM Account];`. The list is strongly typed to the queried sObject type, but it may also be declared as `List<sObject>` when generic handling is needed.

An `sObject` can receive the result of a SOQL query expected to return one record. For example, `Account accountRecord = [SELECT Id, Name FROM Account WHERE Id = :accountId];` assigns the queried record, which is an sObject instance, to a single-record variable. A concrete object such as `Account` is itself an sObject type.

An `Integer` can receive the result of a SOQL aggregate count query. For example, `Integer accountCount = [SELECT COUNT() FROM Account];` returns the number of matching records as an Integer. Salesforce documents both single-record and list query assignment forms, as well as the Integer result of `COUNT()`. See SOQL Queries in Apex and SOQL Aggregate Functions.

QUESTION NO: 44

Which code should be used to update an existing Visualforce page that uses standard

Visualforce components so that the page matches the look and feel of Lightning Experience?

```
Boolean isOK;
integer x;
String theString = 'Hello';

if (isOK == false && theString == 'Hello'){
   x = 1;
} else if (isOK == true && theString == 'Hello'){
   x = 2;
} else if (isOK != null && theString == 'Hello'){
   x = 3;
} else {
   x = 4;
}
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: A

Explanation:

The correct code is the Visualforce page declaration that enables the `lightningStylesheets` attribute, such as `<apex:page lightningStylesheets="true">`. This attribute tells Salesforce to apply Salesforce Lightning Design System-inspired styling to supported standard Visualforce components when the page is displayed in Lightning Experience. It is specifically intended as a low-effort way to modernize an existing page that already uses standard components, avoiding the need to rewrite the page markup with custom SLDS classes or replace the page entirely with a Lightning component.

The attribute should be placed on the root `apex:page` tag. Salesforce then serves the appropriate styling based on the user interface, allowing the same Visualforce page to remain compatible while visually aligning standard inputs, buttons, page blocks, tables, and related components more closely with Lightning Experience. This approach is best suited to pages built primarily from standard Visualforce components, because custom HTML and custom CSS are not automatically converted into Lightning-styled markup. See Salesforce's Visualforce styling for Lightning Experience documentation and the apex:page component reference.

QUESTION NO: 45

Which two characteristics are true for Aura component events?

A. Calling event, stopPropagation ( ) may or may not stop the event propagation based of the current propagation phase.

B. If a container component needs to handle a component event, add a handleFacets=" attribute to Its handler.

C. Only parent components that create subcomponents (either in their markup or programmatically) can handle events.

D. The event propagates to every owner In the containment hierarchy.

ANSWER: A D

Explanation:

Calling `event.stopPropagation()` may have a phase-dependent effect because Aura component events move through the containment hierarchy using capture and bubble phases. During capture, the event is traveling toward the source component; during bubble, it is traveling outward from the source toward containing components. Stopping propagation determines whether the event continues to subsequent eligible components in that lifecycle path, so a developer must consider where the handler runs and which phase is active. This allows event handlers to deliberately control how far a component event travels.

The event propagates to every owner in the containment hierarchy because component events are designed for communication among a source component and the components that contain it. Aura uses this hierarchy to determine which components are eligible to receive and handle the event as it propagates. A component event therefore supports encapsulated communication within a component tree, while still allowing appropriate containing components to react to actions originating in nested components. Salesforce describes component-event propagation, including capture and bubble phases, in its Aura Component Events documentation. See also the Component Event Propagation documentation for the containment-hierarchy model.

QUESTION NO: 46

A recursive transaction is limited by a DML statement creating records for these two objects:

1. Accounts

2. Contacts

The Account trigger hits a stack depth of 16.

Which statement is true regarding the outcome of the transaction?

A. The transaction fails only if the Contact trigger stack depth is greater or equal to 16.

B. The transaction succeeds as long as the Contact trigger stack depth is less than 16.

C. The transaction fails and all the changes are rolled back.

D. The transaction succeeds and all the changes are committed to the database.

ANSWER: D

Explanation:

The transaction succeeds and all the changes are committed to the database. Salesforce permits a maximum recursive trigger depth of 16. Reaching a stack depth of 16 is therefore still within the permitted limit; the platform raises a runtime exception only when execution attempts to go beyond that maximum depth. The limit applies to the recursive trigger execution chain in the transaction, rather than requiring each involved object's trigger to have an independently acceptable depth. Consequently, Account and Contact DML can participate in the same recursion sequence, and the transaction can complete normally when its deepest trigger invocation is exactly 16. Assuming no unrelated validation, exception, governor-limit violation, or additional DML causes another recursive trigger invocation, Salesforce commits the transaction's successful DML work when processing finishes. Salesforce documents recursive trigger depth as a platform limit in its Apex Governor Limits documentation. The platform's Order of Execution documentation also explains that record updates can cause automation and triggers to run again within the same transaction, which is why designs should guard against unintended recursion.