COURSE PROGRESS: 0% COMPLETE

Salesforce Platform Developer I – Practice Test #2

Results

0 of 40 Questions answered correctly

Your time: 00:00:06

You have reached 0 of 40 point(s), (0%)

Restart Quiz View Questions

1 ✕	2 ✕	3 ✕	4 ✕	5 ✕	6 ✕	7 ✕	8 ✕	9 ✕	10 ✕	11 ✕	12 ✕	13 ✕	14 ✕	15 ✕	16 ✕
17 ✕	18 ✕	19 ✕	20 ✕	21 ✕	22 ✕	23 ✕	24 ✕	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40								

When developing Apex code, it's essential to implement robust exception handling to ensure your application can gracefully handle runtime errors. Given this requirement, which of the following approaches correctly implements exception handling in Apex, including the use of custom exceptions when necessary?

Create a custom exception class for every type of exception that might occur in your application to ensure that each exception can be caught and handled specifically.

Live Agent

Correct ai

Use a try-catch-finally block around DML operations, where the catch block handles specific Salesforce exceptions (e.g., DmlException, QueryException), and use custom exceptions for business logic validation errors.

Surround DML operations with a try-catch block that catches generic Exception objects only, to ensure all exceptions are caught and logged.

Avoid using try-catch blocks to keep the code cleaner and rely on Salesforce's built-in error handling mechanisms to automatically roll back transactions in case of errors.

Incorrect

Use a try-catch-finally block around DML operations, where the catch block handles specific Salesforce exceptions (e.g., DmlException, QueryException), and use custom exceptions for business logic validation errors. -> Correct. This approach is best practice as it allows for specific handling of Salesforce-related exceptions and the use of custom exceptions for business logic errors, ensuring that all aspects of exception handling are covered.

Surround DML operations with a try-catch block that catches generic Exception objects only, to ensure all exceptions are caught and logged. -> Incorrect. While catching generic Exception objects can handle all types of exceptions, it's not a best practice to only use generic exceptions as it may not provide specific handling for different error types.

Create a custom exception class for every type of exception that might occur in your application to ensure that each exception can be caught and handled specifically. -> Incorrect. While custom exceptions can be useful for specific error conditions, it's not practical or necessary to create a custom exception for every potential error; instead, use them for unique error conditions not covered by standard exceptions.

Avoid using try-catch blocks to keep the code cleaner and rely on Salesforce's built-in error handling mechanisms to automatically roll back transactions in case of errors. -> Incorrect. Relying on Salesforce's built-in mechanisms without explicit exception handling in your code can lead to unhandled exceptions and make debugging difficult, as it provides no \ to log errors or perform specific actions when an exception occurs.

Live Agent

A Salesforce developer is tasked with automating the process of sending a congratulatory email to the account owner whenever an Opportunity associated with that account is marked as Closed Won. The email should only be sent once per account, the first time any opportunity for that account is marked as Closed Won. Which automation tool or approach is most appropriate to meet this unique requirement?

Scheduled Apex

Validation Rule

Workflow Rule

Process Builder

Apex Trigger **Correct answer**

Incorrect

Apex Trigger -> Correct. An Apex Trigger can efficiently manage the logic needed to determine if the Closed Won opportunity is the first for the account and trigger the email only in that case. This approach allows for the necessary level of control and customization.

Workflow Rule -> Incorrect. While Workflow Rules can trigger email alerts, they lack the capability to check if this is the first Closed Won opportunity for an account without custom fields and additional logic.

Process Builder -> Incorrect. Process Builder provides more flexibility than Workflow Rules and can trigger actions based on record changes, but managing the once-per-account constraint directly would require complex logic and potentially additional fields.

Scheduled Apex -> Incorrect. Scheduled Apex is used for operations that need to run a specific times, not for real-time responses to record changes. It would not be the best

Live Agent

choice for sending immediate congratulatory emails.

Validation Rule -> Incorrect. Validation Rules enforce data integrity and ensure specific conditions are met before records can be saved. They cannot trigger actions like sending emails and do not track historical data, such as whether an email has already been sent.

When developing Lightning Web Components (LWC) in Salesforce, which scenario exemplifies the best practice for utilizing custom events for inter-component communication?

To invoke a method in a grandparent component directly from a grandchild component without using events.

Correct answer

To communicate user-selected options from a child component to its parent component for further processing.

To publish a global event that all components in the application can listen to and react upon.

To pass complex application state changes directly from one sibling component to another.

Incorrect

To communicate user-selected options from a child component to its parent component for further processing. -> Correct. This is a classic use case for custom events in LWC, where a child component needs to communicate up the hierarchy to its parent component, maintaining a clear, decoupled communication path.

To invoke a method in a grandparent component directly from a grandchild component without using events. -> Incorrect. Direct method invocation across multiple levels of component hierarchy without using events bypasses the encapsulation principles of LWC and is not recommended.

Live Agent

To pass complex application state changes directly from one sibling component to another. -> Incorrect. Direct sibling-to-sibling communication is not the intended use of custom events in LWC. Communication should be routed through their mutual parent component using events.

To publish a global event that all components in the application can listen to and react upon. -> Incorrect. LWC does not support the concept of global events in its event model. Communication is intended to be hierarchical or managed via a pub-sub model for cross-component communication.

A Salesforce developer is tasked with implementing an Apex class that will serve as the backend for various types of page components, including Lightning Components, Flow, and Next Best Actions. The class must efficiently manage data retrieval and manipulation tasks, ensuring optimal performance and compatibility across these different frontend interfaces. Considering best practices for Apex development in this scenario, which approach should the developer take?

Exclusively use Future methods to handle all data operations.

Implement bulkified Apex methods to handle large data sets efficiently. **Correct answer**

Utilize synchronous Apex methods for all operations to ensure data consistency.

Use hard-coded SOQL queries within each component-specific Apex method.

Incorrect

Implement bulkified Apex methods to handle large data sets efficiently. -> Correct. Bulkification ensures that the Apex code efficiently processes large sets of data with minimal governor limit consumption, which is crucial for operations triggered by user interfaces like Lightning Components, Flow, and Next Best Actions.

Live Agent

Utilize synchronous Apex methods for all operations to ensure data consistency. -> Incorrect. Relying solely on synchronous methods can lead to performance bottlenecks, especially with complex data operations and UI components that require real-time responsiveness.

Use hard-coded SOQL queries within each component-specific Apex method. -> Incorrect. Hard-coding SOQL queries in component-specific methods reduces the flexibility and maintainability of the code, especially when changes in the data model or business logic occur.

Exclusively use Future methods to handle all data operations. -> Incorrect. While Future methods are useful for handling operations that can be run asynchronously, they are not suitable for all types of operations, especially those requiring immediate data to display in UI components.

The customer service team at Tech Innovations wants an efficient method to track service issues and share solutions documentation with their clients. Which two actions should an administrator take to assist the customer service team in achieving this objective?

- [ ] Utilize Salesforce Knowledge for managing and sharing solutions documentation. Correct answer
- [ ] Implement Service Cloud Console for centralized issue tracking.
- [ ] Use Data Loader to import customer issues as cases.
- [ ] Configure Process Builder to automatically email documentation to customers.
- [ ] Enable Customer Community for direct customer case submission. Correct answer

Incorrect

Live Agent

Utilize Salesforce Knowledge for managing and sharing solutions documentation. -> Correct. Salesforce Knowledge allows the customer service team to create, manage, and share solutions documentation efficiently. This can be a centralized repository for all support documentation, which can be easily shared with clients.

Enable Customer Community for direct customer case submission. -> Correct. Enabling Customer Community gives clients direct access to submit cases, track their progress, and access solutions documentation, fostering a self-service environment.

Implement Service Cloud Console for centralized issue tracking. -> Incorrect. While the Service Cloud Console is an excellent tool for centralized issue tracking, it does not directly facilitate the sharing of solutions documentation with clients.

Configure Process Builder to automatically email documentation to customers. -> Incorrect. Although Process Builder can automate processes like emailing customers, it's not the most efficient method for sharing solutions documentation because it lacks the context-aware sharing capabilities that Salesforce Knowledge and communities provide.

Use Data Loader to import customer issues as cases. -> Incorrect. Using Data Loader to import customer issues as cases is a method for data migration or bulk case creation, not for sharing documentation or enhancing customer engagement directly.

In the Salesforce Platform, which of the following best describes the use of Apex?

To modify the Salesforce data model by adding new objects and fields.

To customize the user interface of the Salesforce Mobile App.

To manage and customize Salesforce reports and dashboards.

To integrate Salesforce with external systems and databases. **Correct answer**

Live Agent

Incorrect

To integrate Salesforce with external systems and databases. -> Correct. Apex can be used to create Web services, perform complex validation over multiple objects, and execute transactional control and logic that spans over multiple operations.

To customize the user interface of the Salesforce Mobile App. -> Incorrect. Apex is primarily used for executing transactional control, business logic, and database operations, not for customizing user interfaces.

To manage and customize Salesforce reports and dashboards. -> Incorrect. Reports and dashboards are typically managed through the Salesforce UI or declarative tools, not Apex.

To modify the Salesforce data model by adding new objects and fields. -> Incorrect. The Salesforce data model is modified through the Object Manager in Setup, not through Apex.

A company wants to track the service history of products sold to customers within Salesforce. Each product can have multiple service records over time, and each service record is related to one specific product. To optimize data management and reporting within Salesforce, how should the developer structure the data model?

Use a Lookup relationship from the Service Record object to the Product object without enforcing referential integrity.

Establish a Many-to-Many relationship between Products and Service Records using a junction object.

Create a Master-Detail relationship between the Product object (Master) and the Service Record object (Detail). Correct answer

Implement an External ID field on the Service Record object to reference the Product object's external system identifier.

Live Agent

Incorrect

Create a Master-Detail relationship between the Product object (Master) and the Service Record object (Detail). -> Correct. A Master-Detail relationship is ideal for this scenario because it tightly links service records to products, allows roll-up summary fields to summarize data from service records on the product, and ensures that deleting a product will also delete its related service records.

Use a Lookup relationship from the Service Record object to the Product object without enforcing referential integrity. -> Incorrect. While a Lookup relationship connects service records to products, not enforcing referential integrity means service records could exist without an associated product, which may not meet the data integrity needs of this scenario.

Implement an External ID field on the Service Record object to reference the Product object's external system identifier. -> Incorrect. While External ID fields are useful for integration purposes, they do not establish a relationship structure between objects within Salesforce needed for tracking service history related to products.

Establish a Many-to-Many relationship between Products and Service Records using a junction object. -> Incorrect. A Many-to-Many relationship is unnecessary and overly complex for this scenario, as each service record is related to one and only one product.

Given a scenario where a Salesforce Developer needs to implement a solution that combines both declarative functionality and Apex code to automate complex business logic efficiently, which of the following approaches would be considered best practices? Select two.

Correct answer

- [ ] Utilize Flow to handle complex logic that requires user interaction and decision-making, calling Apex classes when specific conditions are met for operations that cannot be done declaratively.

- [ ] Implement business logic with workflow rules for all scenarios, including those that require complex data manipulation and branching logic.

Live Agent

Correct answer

- [ ] Use Process Builder to perform simple field updates and call an Apex class for more complex logic that cannot be achieved declaratively.

- [ ] Write all business logic in Apex triggers, regardless of the complexity, to ensure consistency in logic execution.

- [ ] Use a Visualforce page to capture user inputs and then process all business logic in the controller, avoiding the use of declarative tools entirely.

Incorrect

Use Process Builder to perform simple field updates and call an Apex class for more complex logic that cannot be achieved declaratively. -> Correct. This approach leverages the strengths of both declarative and programmatic tools. Process Builder handles straightforward automation, while Apex is used for complex scenarios, maintaining efficiency and clarity.

Utilize Flow to handle complex logic that requires user interaction and decision-making, calling Apex classes when specific conditions are met for operations that cannot be done declaratively. -> Correct. This method combines the powerful user interface and logic capabilities of Flow with the flexibility of Apex for scenarios that require operations beyond declarative limits, offering a balanced approach.

Write all business logic in Apex triggers, regardless of the complexity, to ensure consistency in logic execution. -> Incorrect. Although Apex triggers offer powerful capabilities for custom logic, using them exclusively disregards the benefits of Salesforce's declarative features, which can simplify maintenance and improve efficiency for simpler tasks.

Use a Visualforce page to capture user inputs and then process all business logic in the controller, avoiding the use of declarative tools entirely. -> Incorrect. This approach does not utilize Salesforce's declarative automation capabilities, such as workflow rules or Process Builder, which can lead to unnecessarily complex solutions for scenarios that could be addressed more simply.

Implement business logic with workflow rules for all scenarios, including those that rec complex data manipulation and branching logic. -> Incorrect. Workflow rules are limite

Live Agent

their ability to handle complex logic and data manipulation. Apex should be used for scenarios that exceed declarative tools' capabilities.

In a Salesforce application development scenario aimed at preventing user interface and data access security vulnerabilities, which measure should be taken?

Correct answer

Enforce Field-Level Security (FLS) checks in Apex code to ensure that users can only access fields they have permission to view or edit.

Directly embed user inputs into SOQL queries without sanitization or validation to enhance performance and reduce server load.

Regularly output sensitive data to the browser console for debugging purposes to ensure data integrity and user interface functionality.

Avoid the use of custom data validation mechanisms and rely solely on client-side validation for input fields to streamline user experience.

Incorrect

Enforce Field-Level Security (FLS) checks in Apex code to ensure that users can only access fields they have permission to view or edit. -> Correct. Enforcing Field-Level Security (FLS) checks in Apex code is crucial for maintaining the integrity of data access in Salesforce applications. This practice ensures that Apex classes and triggers respect the field-level permissions set on objects, preventing unauthorized users from viewing or editing fields they do not have access to. This approach aligns with Salesforce's best practices for security, ensuring that data access through the application's user interface adheres to the organization's established security protocols.

Regularly output sensitive data to the browser console for debugging purposes to ensure data integrity and user interface functionality. -> Incorrect. Outputting sensitive data to the browser console poses a significant security risk, exposing potentially confidential information to unauthorized parties.

Live Agent

Directly embed user inputs into SOQL queries without sanitization or validation to enhance performance and reduce server load. -> Incorrect. Embedding user inputs directly into SOQL queries without sanitization can lead to SOQL injection vulnerabilities, allowing attackers to manipulate queries to access or alter data unauthorizedly.

Avoid the use of custom data validation mechanisms and rely solely on client-side validation for input fields to streamline user experience. -> Incorrect. Since relying solely on client-side validation is insecure. Client-side validation can be easily bypassed, necessitating server-side validation to securely enforce input constraints and prevent injection attacks.

Given a scenario where a developer needs to create a Visualforce page to display a list of all active Contacts associated with a specific Account, which approach should the developer take to ensure the page efficiently displays the data using the appropriate controllers or extensions?

Use a standard controller for Account and a custom extension to query active Contacts. **Correct answer**

Implement a Visualforce component without controllers or extensions, using only Apex code embedded within the page.

Utilize a custom controller to handle both the Account and Contact data without extensions.

Use a standard controller for Contacts and ignore the Account relationship.

Incorrect

Use a standard controller for Account and a custom extension to query active Contacts. -> Correct. This approach allows the page to leverage the built-in functionality of the standard controller for Accounts while using a custom extension to specifically query and display active Contacts related to the Account.

Live Agent

Utilize a custom controller to handle both the Account and Contact data without extensions. -> Incorrect. While a custom controller could technically handle the requirement, it may not be as efficient as combining the standard controller for Account with a custom extension, especially for leveraging existing page layouts and actions related to Accounts.

Implement a Visualforce component without controllers or extensions, using only Apex code embedded within the page. -> Incorrect. Embedding Apex code directly in a Visualforce page is not a recommended practice for accessing and manipulating Salesforce data, as it does not follow the MVC (Model-View-Controller) pattern and can lead to maintenance issues.

Use a standard controller for Contacts and ignore the Account relationship. -> Incorrect. This approach does not meet the requirement to display Contacts associated with a specific Account, as it only uses a standard controller for Contacts without considering the Account relationship.

In the context of Salesforce development, when performing unit tests to ensure code coverage and functionality correctness, which of the following best describes the use of the SeeAllData=true annotation?

Correct answer

It allows test methods to access all the data in the organization, bypassing the need to create test data.

It is a best practice to use SeeAllData=true in all test scenarios for comprehensive testing.

It is required on all test classes to access the organization's real data for accurate testing results.

It enhances the performance of test execution by caching data used across multiple test methods.

Live Agent

Incorrect

It allows test methods to access all the data in the organization, bypassing the need to create test data. -> Correct. While SeeAllData=true does allow access to real data, it should be used sparingly due to the potential for unpredictable test outcomes and dependencies on the data's state.

It is required on all test classes to access the organization's real data for accurate testing results. -> Incorrect. Accessing real data in tests is discouraged because it can lead to tests that pass or fail unpredictably depending on the data in the environment.

It enhances the performance of test execution by caching data used across multiple test methods. -> Incorrect. This annotation does not affect performance through caching; it simply provides access to live data, which can actually introduce performance variability.

It is a best practice to use SeeAllData=true in all test scenarios for comprehensive testing. -> Incorrect. The best practice is to create isolated test data using test setup methods. Using real data can make tests less reliable and more dependent on the data's state.

In Salesforce, which two actions can be performed by a Process Builder but not by Workflow Rules? Select two options.

- [ ] Send an outbound message without code.
- [ ] Post to Chatter. Correct answer
- [ ] Evaluate criteria based on formula fields.
- [ ] Create a task and assign it to a user.
- [ ] Update any related record, not just the parent record. Correct answer

Live Agent

Incorrect

Update any related record, not just the parent record. -> Correct. Process Builder allows for updating related records regardless of their relationship to the triggering record. This is a key feature that distinguishes it from Workflow Rules, which can only update the parent record.

Post to Chatter. -> Correct. Process Builder can post messages to Chatter feeds, including on records or to groups. This functionality is not available in Workflow Rules.

Send an outbound message without code. -> Incorrect. Both Workflow Rules and Process Builder can send outbound messages without code. This is a common feature shared between the two.

Evaluate criteria based on formula fields. -> Incorrect. Workflow Rules can also evaluate criteria based on formula fields, making this feature not exclusive to Process Builder.

Create a task and assign it to a user. -> Incorrect. Workflow Rules also have the capability to create tasks and assign them to users, making this an incorrect choice.

What action must a developer take when creating a custom field to ensure it is correctly integrated into the Salesforce org's data model?

Update the field-level security to make the field visible to the appropriate profiles. Correct answer

Create a new record type for the custom field.

Assign the field to all page layouts.

Automatically populate the field on all existing records.

Incorrect

Live Agent

Update the field-level security to make the field visible to the appropriate profiles. -> Correct. Setting field-level security is essential to ensure that the right users have access to the new custom field according to their profiles.

Automatically populate the field on all existing records. -> Incorrect. While you can set default values for new records, automatically populating the field on existing records requires either a mass update operation or specifying a default value, if applicable, but is not a required step in field creation.

Assign the field to all page layouts. -> Incorrect. While adding the field to page layouts is a common step, it's not mandatory for all layouts unless it's necessary for business processes.

Create a new record type for the custom field. -> Incorrect. Creating a new record type is not a required action for integrating a custom field. Record types are used to offer different business processes, picklist values, and page layouts to different users, not for field integration.

Which two statements accurately describe key concepts related to Salesforce's multi-tenant architecture and the Lightning Component Framework, in the context of MVC (Model-View-Controller) architecture and component-based development? Select two correct answers.

Correct answer

- [ ] Lightning Web Components (LWC) leverage web standards, and they work alongside Aura Components to build efficient, scalable user interfaces.
- [ ] The Lightning Component Framework is designed to support only client-side rendering, making server-side logic unnecessary.
- [ ] In Salesforce MVC architecture, the View can directly modify data in the Model without going through a Controller, for simplicity.
- [ ] Aura Components are primarily intended for back-end data processing, similar to Apex classes, within the Lightning Component Framework.

Live Agent

Correct ai

- [ ] Components in the Lightning Component Framework should avoid calling Apex controllers synchronously to prevent blocking the user interface.

Incorrect

Components in the Lightning Component Framework should avoid calling Apex controllers synchronously to prevent blocking the user interface. -> Correct. Making asynchronous calls to Apex controllers is a best practice to ensure that the UI remains responsive. Synchronous calls can block the UI, leading to a poor user experience.

Lightning Web Components (LWC) leverage web standards, and they work alongside Aura Components to build efficient, scalable user interfaces. -> Correct. LWC is a modern framework that promotes the use of standard web technologies. It can coexist with Aura Components, allowing developers to build efficient and scalable applications on the Salesforce platform.

In Salesforce MVC architecture, the View can directly modify data in the Model without going through a Controller, for simplicity. -> Incorrect. This statement is incorrect because, in MVC architecture, the View interacts with the Model through the Controller to maintain separation of concerns and ensure data integrity.

The Lightning Component Framework is designed to support only client-side rendering, making server-side logic unnecessary. -> Incorrect. While the Lightning Component Framework emphasizes client-side rendering for performance and responsiveness, server-side logic via Apex is crucial for data manipulation, business logic execution, and enforcing security.

Aura Components are primarily intended for back-end data processing, similar to Apex classes, within the Lightning Component Framework. -> Incorrect. Aura Components, like Lightning Web Components, are used for building user interfaces in the Lightning Component Framework. Apex classes are used for back-end data processing and business logic.

Live Agent

A Salesforce Platform Developer is working on enhancing the security measures of a Salesforce application's user interface and its data access layers to mitigate potential vulnerabilities. Considering the best practices for securing a Salesforce application, which of the following measures should the developer implement to most effectively prevent user interface and data access security vulnerabilities?

Disable Salesforce's built-in security features, such as XSS protection and CSRF tokens, to gain more control over the application's security checks and custom implementations.

Embed user inputs directly into SOQL queries without sanitization to ensure dynamic and responsive data retrieval based on user actions.

Hardcode sensitive information like access tokens and credentials in Lightning Web Components for quicker data access and to streamline the authentication process.

Correct answer

Ensure that all Apex classes adhere to the principle of least privilege by enforcing Field-Level Security (FLS), Object-Level Security (OLS), and sharing rules, even if Salesforce enforces these at the UI level.

Incorrect

Ensure that all Apex classes adhere to the principle of least privilege by enforcing Field-Level Security (FLS), Object-Level Security (OLS), and sharing rules, even if Salesforce enforces these at the UI level. -> Correct. Enforcing FLS, OLS, and sharing rules in Apex classes ensures that custom code respects the organization's security model, effectively preventing unauthorized data access and ensuring consistent security enforcement across both UI and data layers.

Embed user inputs directly into SOQL queries without sanitization to ensure dynamic and responsive data retrieval based on user actions. -> Incorrect. Directly embedding user inputs into SOQL queries without sanitization can lead to SOQL injection attacks, exposing the application to unauthorized data access.

Live Agent

Hardcode sensitive information like access tokens and credentials in Lightning Web Components for quicker data access and to streamline the authentication process. ->

Incorrect. Hardcoding sensitive information in client-side code like Lightning Web Components exposes the application to security risks, including unauthorized access and data breaches.

Disable Salesforce's built-in security features, such as XSS protection and CSRF tokens, to gain more control over the application's security checks and custom implementations. -> Incorrect. Disabling built-in security features like XSS protection and CSRF tokens removes critical safeguards against common web vulnerabilities, making the application more susceptible to attacks.

Consider the following question about the User Interface, focusing on the description of the Lightning Component framework, its benefits, and the types of content that can be included in a Lightning web component. Which two statements accurately describe the Lightning Component framework, including its benefits and the types of content that can be included in a Lightning web component?

- [ ] The framework is designed to enhance the user experience by utilizing only traditional web technologies such as HTML and CSS, excluding modern JavaScript.

- [ ] One of the benefits of the Lightning Component framework is its backward compatibility with Aura components, ensuring seamless integration within Salesforce applications.

Correct answer

- [ ] Lightning web components can only execute client-side scripts and cannot interact with Salesforce data directly.

- [ ] Lightning web components are limited to static HTML content and cannot include dynamic content such as JavaScript or CSS.

- [ ] A key benefit of the Lightning Component framework is its compliance with web standards, making it easier for developers to create components that are portable and interoperable across different web environments.

Correct answer

Live Agent

Incorrect

One of the benefits of the Lightning Component framework is its backward compatibility with Aura components, ensuring seamless integration within Salesforce applications. -> Correct. The framework provides backward compatibility with Aura components, allowing developers to use both Aura and Lightning web components together in Salesforce applications.

A key benefit of the Lightning Component framework is its compliance with web standards, making it easier for developers to create components that are portable and interoperable across different web environments. -> Correct. The framework is built with web standards in mind, ensuring that components are portable and can operate seamlessly across various web environments.

Lightning web components can only execute client-side scripts and cannot interact with Salesforce data directly. -> Incorrect. Lightning web components can execute both client-side and server-side scripts, allowing them to interact directly with Salesforce data through Apex classes.

The framework is designed to enhance the user experience by utilizing only traditional web technologies such as HTML and CSS, excluding modern JavaScript. -> Incorrect. The Lightning Component framework uses modern web technologies, including HTML, CSS, and modern JavaScript, to enhance the user experience and developer productivity.

Lightning web components are limited to static HTML content and cannot include dynamic content such as JavaScript or CSS. -> Incorrect. Lightning web components support dynamic content, including JavaScript and CSS, enabling developers to create rich, interactive user interfaces.

In the context of Apex transactions and the Salesforce save order of execution, which statement accurately describes how to prevent recursion and/or cascading effects in triggers?

Rely exclusively on workflow rules and process builder for automation, as these toc not suffer from recursion issues.

Live Agent

Correct answer

Implement trigger frameworks or handler classes with static variables to track and prevent re-execution of logic within the same transaction.

Always use after insert and after update triggers to ensure that all recursion is automatically managed by the platform.

Configure all triggers to execute asynchronously using future methods, thereby isolating operations and preventing recursion.

Incorrect

Implement trigger frameworks or handler classes with static variables to track and prevent re-execution of logic within the same transaction. -> Correct. Implementing a trigger framework or using handler classes with static variables to mark when a trigger has already run for specific records in a transaction is a common pattern to prevent recursion. This approach allows developers to control and prevent unnecessary re-execution of logic.

Always use after insert and after update triggers to ensure that all recursion is automatically managed by the platform. -> Incorrect. Using after insert and after update triggers does not automatically manage recursion. Recursion management must be explicitly implemented to prevent triggers from executing logic multiple times for the same records within a transaction.

Rely exclusively on workflow rules and process builder for automation, as these tools do not suffer from recursion issues. -> Incorrect. While workflow rules and Process Builder are declarative tools that can reduce the complexity of automation, they can still cause recursion issues when used alongside triggers without proper design and control mechanisms in place.

Configure all triggers to execute asynchronously using future methods, thereby isolating operations and preventing recursion. -> Incorrect. Executing all triggers asynchronously using future methods might avoid some forms of recursion but can introduce other complexities and limitations, such as the inability to make callouts after DML operations in the same transaction and governor limits on future methods.

Live Agent

A developer needs to ensure that an Apex class can only be executed by certain profiles within the Salesforce org. Which approach should the developer take to enforce this requirement?

Use the with sharing keyword on the Apex class to automatically enforce the security settings of the running user's profile.

Assign the Apex class to a Permission Set that is only assigned to the allowed profiles.

Mark the Apex class as private to restrict its execution to other Apex code within the same application namespace.

Correct answer

Implement custom logic within the Apex class that checks the ProfileId of the UserInfo class against a list of allowed profile IDs.

Incorrect

Implement custom logic within the Apex class that checks the ProfileId of the UserInfo class against a list of allowed profile IDs. -> Correct. This approach allows the developer to programmatically control access to the Apex class by checking the running user's profile against a predefined list of authorized profiles.

Use the with sharing keyword on the Apex class to automatically enforce the security settings of the running user's profile. -> Incorrect. The with sharing keyword enforces record-level access based on the running user's permissions and does not restrict execution based on user profile.

Mark the Apex class as private to restrict its execution to other Apex code within the same application namespace. -> Incorrect. The private keyword in Apex is used to restrict visibility of class members to the same class and does not control execution access based on user profiles.

Assign the Apex class to a Permission Set that is only assigned to the allowed profiles. -> Incorrect. Permission Sets extend users' privileges but do not provide a method to programmatically check and restrict class execution within the Apex code itself.

Live Agent

In a scenario where an organization wants to automatically send a welcome email to a new Contact when it is created, and track the email send status on the Contact record, which best practices for writing Apex classes and triggers should be followed to implement this functionality?

- [ ] Create a before insert trigger on the Contact object that checks if an email has been sent and updates the email status field, relying on a separate scheduled Apex job to send the email.
- [ ] Implement the email sending logic directly in the before insert trigger on the Contact object, updating the email status field immediately after sending the email.
- [ ] Use an after insert trigger on the Contact object to call a batch Apex class that sends the welcome email and updates the Contact record with the email send status. Correct answer
- [ ] Utilize a future method invoked by an after insert trigger on the Contact object to send the welcome email and update the email status, ensuring the process is asynchronous. Correct answer
- [ ] Write an after insert trigger on the Contact object that immediately sends the email and updates the Contact record's email status field within the same transaction.

Incorrect

Use an after insert trigger on the Contact object to call a batch Apex class that sends the welcome email and updates the Contact record with the email send status. -> Correct. This approach separates concerns by using a trigger to initiate the process and batch Apex to handle the processing, allowing for efficient handling of bulk operations and asynchronous processing.

Utilize a future method invoked by an after insert trigger on the Contact object to send the welcome email and update the email status, ensuring the process is asynchronous. -> Correct. This approach allows for the asynchronous execution of operations that are not critical to the immediate transaction, like sending emails, and helps manage governor li

Live Agent

Implement the email sending logic directly in the before insert trigger on the Contact object, updating the email status field immediately after sending the email. -> Incorrect. Directly implementing complex logic and external actions (like sending emails) in triggers is not recommended due to potential for governor limit issues and the complexity of error handling.

Create a before insert trigger on the Contact object that checks if an email has been sent and updates the email status field, relying on a separate scheduled Apex job to send the email. -> Incorrect. Using a before insert trigger to update fields based on actions that haven't yet occurred (sending an email) does not align with logical execution order or best practices.

Write an after insert trigger on the Contact object that immediately sends the email and updates the Contact record's email status field within the same transaction. -> Incorrect. While technically feasible, performing external calls (like sending emails) and subsequent updates within the same transaction in an after trigger can lead to complications, including handling of exceptions and governor limits.

When deploying a new Apex trigger and its associated test classes from a sandbox environment to production, what is the most efficient way to ensure the deployment will succeed in terms of code coverage and test success?

Correct answer

Execute all relevant tests in the sandbox environment before deployment to validate code coverage and discover any issues.

Manually review code and perform peer reviews, skipping automated tests for quicker deployment.

Use the 'Run All Tests' option in production post-deployment to confirm code coverage.

Deploy the trigger and test classes together without running tests in the sandbox.

Live Agent

Incorrect

Execute all relevant tests in the sandbox environment before deployment to validate code coverage and discover any issues. -> Correct. Executing all relevant tests in the sandbox before deployment is the most efficient way to ensure that the deployment will likely succeed by identifying and resolving any issues beforehand.

Deploy the trigger and test classes together without running tests in the sandbox. -> Incorrect. Deploying without running tests in the sandbox misses the opportunity to identify and resolve issues before they reach production.

Use the 'Run All Tests' option in production post-deployment to confirm code coverage. -> Incorrect. Relying on post-deployment tests in production can lead to deployment failures if the code does not meet the coverage and test success criteria.

Manually review code and perform peer reviews, skipping automated tests for quicker deployment. -> Incorrect. Manual reviews cannot substitute for automated tests, which are essential for verifying code functionality and coverage across various scenarios.

What is the correct way to describe the use of SOQL (Salesforce Object Query Language) in Salesforce development?

SOQL is used to update existing records in Salesforce databases.

SOQL is used to insert new records into the Salesforce database.

SOQL is used to query records from the Salesforce database based on specific criteria. Correct answer

SOQL is used to create new Salesforce objects and fields.

Incorrect

Live Agent

SOQL is used to query records from the Salesforce database based on specific criteria. -> Correct. SOQL is used to retrieve data from the Salesforce database. It allows developers to specify the criteria that determine which records to select.

SOQL is used to insert new records into the Salesforce database. -> Incorrect. SOQL is specifically designed for querying data. To insert new records, DML operations such as insert are used in Apex.

SOQL is used to update existing records in Salesforce databases. -> Incorrect. SOQL is for querying data, not updating it. DML operations such as update are used for modifying existing records.

SOQL is used to create new Salesforce objects and fields. -> Incorrect. SOQL is not used for creating objects or fields; it is only for querying data. Salesforce's Schema Builder or the Setup menu can be used for creating objects and fields.

In a complex sales process, a sales team needs to automate a discount approval workflow. When an Opportunity reaches a certain discount threshold (above 15%), it must automatically send an email to a manager for approval and create a Task for follow-up. Additionally, if the discount is approved, a custom object record called DiscountApproval__c should be created to track the approval. The solution should leverage both declarative functionality and Apex. How can this requirement be best implemented?

Use a Workflow Rule for the email alert and Task creation, and an Apex Trigger to create the DiscountApproval__c record.

Use a Lightning Component to manually trigger the process, invoking an Apex class that handles all operations.

Correct answer

Use a Process Builder to send the email, create the Task, and call an Apex class to handle the creation of the DiscountApproval__c record.

Live Agent

Use an Apex Trigger for the entire process, including sending emails, creating Tasks, and creating DiscountApproval__c records.

Incorrect

Use a Process Builder to send the email, create the Task, and call an Apex class to handle the creation of the DiscountApproval__c record. -> Correct. This approach uses Process Builder for declarative automation and calls Apex for the custom logic, meeting the requirement to use both declarative functionality and Apex.

Use a Workflow Rule for the email alert and Task creation, and an Apex Trigger to create the DiscountApproval__c record. -> Incorrect. Workflow Rules cannot conditionally trigger Apex.

Use an Apex Trigger for the entire process, including sending emails, creating Tasks, and creating DiscountApproval__c records. -> Incorrect. While possible, this approach does not utilize declarative automation, which was a specific requirement.

Use a Lightning Component to manually trigger the process, invoking an Apex class that handles all operations. -> Incorrect. This approach requires manual intervention and does not leverage declarative automation, missing the requirement for an automated solution.

Global Retail Inc. operates with a private sharing model for the Contact object. The marketing, sales, and customer service teams all use the Standard User profile. A project requires that sales representatives have read/write access, while marketing and customer service teams need read-only access to specific contact records related to a new product launch. What should the Administrator do to ensure that each team has the appropriate access?

Use manual sharing to individually share each contact record with the appropriate teams.

Create a custom profile for each team with specific object permissions.

Live Agent

Modify the Standard User profile to grant read/write access to the sales team.

Create a public group for each team and use sharing rules to grant access based on group membership.

Correct answer

Incorrect

Create a public group for each team and use sharing rules to grant access based on group membership. -> Correct. Sharing rules can be used to extend sharing access beyond the org-wide defaults by specifying groups of users and the level of access.

Modify the Standard User profile to grant read/write access to the sales team. -> Incorrect. Modifying the Standard User profile would affect all users with that profile, not just the sales representatives.

Use manual sharing to individually share each contact record with the appropriate teams. -> Incorrect. While manual sharing can adjust access to individual records, it is not scalable or efficient for groups of users or large numbers of records.

Create a custom profile for each team with specific object permissions. -> Incorrect. Creating a custom profile for each team would provide broad access to the Contact object but would not address the need for record-level access control based on the project requirements.

A Salesforce developer encounters an issue where a batch job is failing intermittently, and certain automated processes and flows are not executing as expected. To identify and resolve these issues, the developer needs to employ effective debugging and monitoring strategies across these asynchronous operations and workflows. Which of the following approaches should the developer take to efficiently debug and monitor the system issues?

Rely solely on the Salesforce Setup Audit Trail for monitoring and debugging all asynchronous processes and batch jobs.

Live Agent

Exclusively use email notifications within Apex code and process builders for error handling and debugging.

Deploy the application to a new sandbox environment for each debugging session to isolate and identify the issues.

Use the Developer Console's Debug Logs to monitor real-time execution of code and process automation, setting specific user trace flags.

Correct answer

Incorrect

Use the Developer Console's Debug Logs to monitor real-time execution of code and process automation, setting specific user trace flags. -> Correct. The Developer Console's Debug Logs, along with setting user trace flags, allow developers to monitor the real-time execution of code, including batch jobs, and automated processes. This approach helps in identifying and debugging issues by providing detailed execution logs.

Rely solely on the Salesforce Setup Audit Trail for monitoring and debugging all asynchronous processes and batch jobs. -> Incorrect. While the Setup Audit Trail is useful for tracking changes made in the Salesforce setup, it does not provide detailed execution logs or error information for debugging asynchronous processes and batch jobs.

Deploy the application to a new sandbox environment for each debugging session to isolate and identify the issues. -> Incorrect. While sandbox environments are useful for testing and debugging, deploying the application to a new sandbox for each debugging session is not practical or efficient for monitoring and debugging ongoing issues.

Exclusively use email notifications within Apex code and process builders for error handling and debugging. -> Incorrect. While email notifications can alert developers to issues, relying solely on them for debugging is not effective. They do not provide the detailed context or execution logs necessary for thorough debugging.

Live Agent

In Salesforce, what allows developers to execute asynchronous Apex code?

Validation Rules

Future Methods

Correct answer

Process Builder

Workflow Rules

Incorrect

Future Methods -> Correct. Future methods are used to run specific Apex code asynchronously, allowing for operations that take a long time to complete without holding up the execution of your main code.

Workflow Rules -> Incorrect. Workflow rules automate standard internal procedures and processes but cannot execute Apex code.

Process Builder -> Incorrect. Process Builder automates various actions within Salesforce but does not support the execution of asynchronous Apex code directly.

Validation Rules -> Incorrect. Validation rules enforce data integrity and accuracy but cannot execute any Apex code, synchronous or asynchronous.

A Salesforce developer needs to automate a complex approval process for a custom object Project__c. The process requires multi-step validations against various criteria, some of which involve complex logic that cannot be handled declaratively. Once a Project__c record meets initial criteria, it should automatically enter an approval process. If the project has a specific type, additional Apex-based validations are required before it can be approved. Which approach best combines declarative functionality and Apex to fulfill these requirements?

Live Agent

Create a Process Builder process to automatically submit Project__c records for approval, and use an Apex trigger for all validation logic.

Configure a Process Builder to handle initial criteria checks and call an Invocable Method for complex Apex validations before submitting for approval.

Correct answer

Utilize a before-save Flow to handle all validations and directly submit records for approval if they pass.

Implement an Apex trigger to perform initial validations and use a Process Builder to submit Project__c records for approval if they meet the criteria.

Incorrect

Configure a Process Builder to handle initial criteria checks and call an Invocable Method for complex Apex validations before submitting for approval. -> Correct. This method smartly uses Process Builder for initial, simpler validation checks that can be done declaratively, and leverages Apex for complex logic via an Invocable Method. Only records passing both stages are submitted for approval, efficiently combining declarative and programmatic capabilities.

Create a Process Builder process to automatically submit Project__c records for approval, and use an Apex trigger for all validation logic. -> Incorrect. While Process Builder can automate the submission for approval, using an Apex trigger exclusively for validation might not be the most efficient use of resources, especially for validations that could be declaratively managed.

Utilize a before-save Flow to handle all validations and directly submit records for approval if they pass. -> Incorrect. A before-save Flow is efficient for field updates before record saving but lacks the capability to submit records for Salesforce approval processes directly. This approach also doesn't effectively separate complex Apex-based validations from simpler declarative checks.

Implement an Apex trigger to perform initial validations and use a Process Builder to submit Project__c records for approval if they meet the criteria. -> Incorrect. This appr

Live Agent

inversely prioritizes Apex for initial validations, which may not be the most effective strategy, particularly for criteria that can be assessed declaratively. It does not leverage the platform's strengths optimally.

Cloud Services Inc. wants to automatically update the "Customer Satisfaction Score" field on a Case record to "High" if the case is closed within 2 days of its creation. The "Customer Satisfaction Score" is a custom picklist field on the Case object. An Email notification should also be sent to the case owner when this update occurs. What type of workflow action is required to fulfill this requirement?

Time-Based Workflow Action **Correct answer**

Field Update Workflow Action

Rule-Based Workflow Action

Immediate Workflow Action

Incorrect

Time-Based Workflow Action -> Correct. This action allows the workflow to wait for a specific condition to be met related to time, such as a case being closed within 2 days of creation, before executing actions like field updates or email alerts.

Immediate Workflow Action -> Incorrect. Immediate actions occur right when a record is created or edited and do not account for the passage of time between events.

Field Update Workflow Action -> Incorrect. While a field update is part of the requirement, the specific need to wait until a case is closed within 2 days necessitates a time-based trigger.

Rule-Based Workflow Action -> Incorrect. This term is not specifically used in Salesfor workflow terminology. Workflows are indeed triggered by rules, but the options to exe

Live Agent

based on time are specifically categorized as "Time-Based Workflow Actions."

Which practices correctly identify how and when to use Salesforce Developer tools such as Salesforce DX, Salesforce CLI, and Developer Console? Select two correct answers.

- [ ] Apply Salesforce CLI exclusively for data manipulation tasks such as inserting, updating, or deleting records in the Salesforce database.
- [ ] Rely on the Developer Console for comprehensive version control management and collaborative development across teams.
- [ ] Use Salesforce DX to establish a source-driven development environment, enabling better team collaboration through version control systems and facilitating modular development and application packaging. Correct answer
- [ ] Utilize Salesforce CLI for quick retrieval and deployment of metadata between Salesforce environments, facilitating continuous integration and continuous deployment (CI/CD) practices. Correct answer

Incorrect

Utilize Salesforce CLI for quick retrieval and deployment of metadata between Salesforce environments, facilitating continuous integration and continuous deployment (CI/CD) practices. -> Correct. Salesforce CLI is a command-line tool that streamlines the retrieval and deployment of metadata, making it ideal for implementing CI/CD practices by automating these processes across different environments.

Use Salesforce DX to establish a source-driven development environment, enabling better team collaboration through version control systems and facilitating modular development and application packaging. -> Correct. Salesforce DX supports source-driven development, allowing teams to use version control systems for collaboration, and makes modular development and application packaging more manageable, promoting best practices ir software development lifecycle management.

Live Agent

Rely on the Developer Console for comprehensive version control management and collaborative development across teams. -> Incorrect. The Developer Console provides a web-based interface for code editing, debugging, and performance analysis but lacks comprehensive version control and collaborative development features. These tasks are better managed through Salesforce DX and associated version control systems.

Apply Salesforce CLI exclusively for data manipulation tasks such as inserting, updating, or deleting records in the Salesforce database. -> Incorrect. While Salesforce CLI can perform data manipulation tasks, its use is not exclusive to these operations. It's also heavily utilized for automation, scripting, metadata management, and facilitating DevOps processes.

In an Apex class, you are tasked with implementing logic that iterates over a list of Account records and performs an action only if the Account's AnnualRevenue is greater than 1 million. Which control flow statement and structure correctly implements this requirement?

for(Account acc : listOfAccounts) { while(acc.AnnualRevenue > 1000000) { // Perform action } }

Correct answer

for(Account acc : listOfAccounts) { if(acc.AnnualRevenue > 1000000) { // Perform action } }

for(int i = 0; i 1000000) { // Perform action } }

if(listOfAccounts.size() > 0) { // Perform action }

Incorrect

```
for(Account acc : listOfAccounts) {
if(acc.AnnualRevenue > 1000000) {
// Perform action
}
}
```

Live Agent

This structure uses a for loop to iterate over each Account in listOfAccounts and an if statement to check if AnnualRevenue is greater than 1 million. If the condition is true, it performs the specified action.

Incorrect answers:

```
if(listOfAccounts.size() > 0) {
// Perform action
}
```

This structure only checks if the list of accounts is not empty and does not iterate over the accounts or check each Account's AnnualRevenue.

```
for(Account acc : listOfAccounts) {
while(acc.AnnualRevenue > 1000000) {
// Perform action
}
}
```

Using a while loop inside a for loop in this context is incorrect because it could result in an infinite loop if an Account's AnnualRevenue is greater than 1 million, as the condition does not change within the while loop.

```
for(int i = 0; i 1000000) {
// Perform action
}
}
```

This structure attempts to use array access syntax listOfAccounts[i] which is incorrect for accessing elements of a List in Apex. The correct method is listOfAccounts.get(i).

Live Agent

Given a scenario where you are developing a custom Apex solution to automate business processes involving multiple objects in Salesforce, which of the following practices should you follow to ensure your Apex classes and triggers are scalable, maintainable, and compliant with Salesforce's best practices?

Place all business logic directly inside each trigger to ensure that it is executed with every DML operation, avoiding the use of helper classes or services.

Avoid writing test classes for triggers and helper classes, assuming that manual testing will suffice for verifying business logic.

Correct answer

Utilize a single trigger per object that delegates business logic to handler classes, where different methods are used for different DML contexts (before insert, after insert, etc.).

Develop a large number of small, object-specific triggers for each DML operation (insert, update, delete) to handle business logic separately.

Incorrect

Utilize a single trigger per object that delegates business logic to handler classes, where different methods are used for different DML contexts (before insert, after insert, etc.). -> Correct. This practice ensures better organization of code, makes it easier to maintain and test, and helps in managing the execution context and governor limits more effectively.

Develop a large number of small, object-specific triggers for each DML operation (insert, update, delete) to handle business logic separately. -> Incorrect. Having a large number of small triggers for each DML operation can lead to maintenance challenges and difficulties in ensuring the correct execution order of business logic.

Place all business logic directly inside each trigger to ensure that it is executed with every DML operation, avoiding the use of helper classes or services. -> Incorrect. Placing all business logic directly inside triggers makes the code less maintainable and more difficult to test. It can also lead to issues with governor limits.

Live Agent

Avoid writing test classes for triggers and helper classes, assuming that manual testing will suffice for verifying business logic. -> Incorrect. Salesforce requires a minimum of 75% code coverage for deployments to production environments. Automated test classes are essential for ensuring the reliability of your code and meeting deployment requirements.

When encountering a performance issue with a Visualforce page that dynamically interacts with large data sets, which strategy should a Salesforce developer employ to pinpoint and address the root cause of the slowdown?

Rely on Salesforce's built-in optimization for Visualforce pages without further investigation.

Correct answer

Review the Visualforce page's Apex controllers and extensions using the Developer Console's Execution Overview to identify inefficient SOQL queries.

Utilize Workbench to perform SOQL queries manually and verify the response time outside of the Visualforce page.

Increase the SOQL query limit in the Salesforce org settings to accommodate the large data sets.

Incorrect

Review the Visualforce page’s Apex controllers and extensions using the Developer Console’s Execution Overview to identify inefficient SOQL queries. -> Correct. The Developer Console’s Execution Overview can help identify inefficient SOQL queries and other performance bottlenecks in Apex controllers and extensions used by the Visualforce page.

Increase the SOQL query limit in the Salesforce org settings to accommodate the large data sets. -> Incorrect. Increasing SOQL query limits is not possible as they are enforced by Salesforce to ensure platform performance and cannot be adjusted to solve performar issues.

Live Agent

Utilize Workbench to perform SOQL queries manually and verify the response time outside of the Visualforce page. -> Incorrect. While testing query performance in Workbench can provide insights, it does not directly address inefficiencies within the Visualforce page or its controller logic.

Rely on Salesforce’s built-in optimization for Visualforce pages without further investigation. -> Incorrect. Trusting Salesforce’s built-in optimizations alone overlooks the necessity of analyzing and optimizing custom code and queries for specific performance issues.

A Salesforce Platform Developer is tasked with automating a complex business process that involves multiple steps, including updating records, sending email notifications, and creating tasks when a specific condition is met on an Opportunity record. The process needs to be efficient, easily maintainable, and scalable. Which process automation tool should the developer use to meet these requirements?

Process Builder

Workflow Rules

Lightning Web Components (LWC)

Apex Triggers **Correct answer**

Incorrect

Apex Triggers -> Correct. Apex Triggers offer the greatest flexibility and control, allowing developers to handle complex logic, perform bulk operations efficiently, and ensure scalability and maintainability of the business process automation.

Workflow Rules -> Incorrect. Workflow Rules are limited in their ability to handle com multi-step processes and cannot create tasks based on multiple conditions or sequenc

Live Agent

events.

Process Builder -> Incorrect. While Process Builder can handle multi-step processes and more complex conditions than Workflow Rules, it may not offer the best performance or maintainability for highly complex scenarios, especially at scale.

Lightning Web Components (LWC) -> Incorrect. LWC are used for creating custom user interfaces and enhancing the user experience, not for automating backend business processes.

In a Salesforce application, an IT developer needs to implement a custom feature that dynamically interacts with various page components, including Lightning Components and Flow, to provide users with personalized Next Best Actions based on their current context within the application. Which two Apex implementations would be most effective for achieving this objective?

- [ ] Use Apex to generate Visualforce pages dynamically for each user action, embedding Lightning Components and Flow within them.
- [ ] Create Apex Scheduled Jobs to periodically update Next Best Actions for all users, regardless of their current context.
- [ ] Utilize Platform Events in Apex to publish user actions and subscribe to these events in Lightning Components for real-time updates. Correct answer
- [ ] Use Apex to call AuraEnabled methods and interact with Lightning Components directly, passing user context as parameters. Correct answer
- [ ] Implement Apex Triggers to automatically update Flow variables based on changes in Salesforce records.

Incorrect

Live Agent

Use Apex to call AuraEnabled methods and interact with Lightning Components directly, passing user context as parameters. -> Correct. This approach is correct because AuraEnabled methods in Apex allow server-side logic to be exposed to client-side Lightning Components, enabling dynamic interaction based on user context.

Utilize Platform Events in Apex to publish user actions and subscribe to these events in Lightning Components for real-time updates. -> Correct. Platform Events provide a powerful way to communicate changes and user actions between Apex and Lightning Components, enabling dynamic UI updates and personalized user interactions.

Implement Apex Triggers to automatically update Flow variables based on changes in Salesforce records. -> Incorrect. While Apex Triggers can automate processes based on record changes, they do not directly interact with Flow variables for real-time user interactions in the UI.

Create Apex Scheduled Jobs to periodically update Next Best Actions for all users, regardless of their current context. -> Incorrect. Scheduled Jobs are used for batch processing at defined intervals and are not suitable for real-time, context-sensitive UI interactions.

Use Apex to generate Visualforce pages dynamically for each user action, embedding Lightning Components and Flow within them. -> Incorrect. While embedding Lightning Components and Flow in Visualforce pages is possible, this approach does not leverage Apex for real-time, dynamic interaction with page components based on user context.

Given a complex requirement to automatically notify a Sales Manager via email when any Opportunity associated with their team's Accounts has been closed without winning for more than 30 days, and ensure this process adheres to Apex best practices for process automation and logic. Considering the need for scalable, maintainable code and efficient processing in bulk operations, which of the following solutions is most appropriate?

Utilize Process Builder to send an email alert when an Opportunity meets the criteri

Live Agent

Correct a

Create a scheduled Apex class that utilizes a custom email service to manage email notifications based on queried Opportunities.

Develop a trigger on the Opportunity object that sends an email when the criteria are met.

Write a batch Apex class that queries Opportunities daily, checks the criteria, and sends emails directly.

Incorrect

Create a scheduled Apex class that utilizes a custom email service to manage email notifications based on queried Opportunities. -> Correct. This approach adheres to best practices by separating concerns (querying data and sending emails are handled distinctly), allows for efficient bulk processing, and leverages custom services for flexibility in managing email notifications.

Write a batch Apex class that queries Opportunities daily, checks the criteria, and sends emails directly. -> Incorrect. Batch Apex is suitable for processing large data volumes and can be scheduled daily, but directly sending emails from Apex is not a best practice due to potential limits on email sends and lack of flexibility in managing email templates.

Utilize Process Builder to send an email alert when an Opportunity meets the criteria. -> Incorrect. Process Builder allows for automation without code and can send email alerts, but it might not efficiently handle the complexity of checking Opportunities over a 30-day period or ensure bulk data processing efficiency.

Develop a trigger on the Opportunity object that sends an email when the criteria are met. -> Incorrect. Implementing logic directly in a trigger to send emails can lead to scalability issues, governor limit concerns, and challenges with bulk data operations, which contradicts best practices for trigger development.

Given a scenario where you need to implement logic that iterates over a list of records and performs different actions based on specific conditions, which two of the following Apex control flow statements would be most appropriate to use and apply?

- [ ] The switch statement to selectively execute code blocks based on the value of a variable or expression. Correct answer
- [ ] The for loop to iterate over each record in the list, applying conditions within the loop to perform different actions. Correct answer
- [ ] The break statement to immediately exit a loop when a specific condition is met, regardless of whether there are remaining items to iterate over.
- [ ] The goto statement to jump to different parts of the code based on conditions.
- [ ] The continue statement to skip the current iteration of a loop and proceed with the next iteration if a certain condition is met.

Incorrect

The switch statement to selectively execute code blocks based on the value of a variable or expression. -> Correct. The switch statement in Apex allows for clean and readable conditional logic, enabling the execution of different code blocks based on the value of a variable or expression. It's particularly useful for handling multiple conditions without requiring complex nested if-else structures.

The for loop to iterate over each record in the list, applying conditions within the loop to perform different actions. -> Correct. The for loop is ideal for iterating over a list of records, allowing for the application of conditions within each iteration to perform actions based on specific criteria. It provides a structured way to traverse a collection and apply logic to each element.

The goto statement to jump to different parts of the code based on conditions. -> Incor

Apex does not support the goto statement. Control flow in Apex is managed using

Live Agent

structured programming constructs like loops (for, while, do-while) and conditional statements (if-else, switch), rather than unstructured jump statements.

The continue statement to skip the current iteration of a loop and proceed with the next iteration if a certain condition is met. -> Incorrect. While the continue statement is used within loops to skip to the next iteration, it doesn't directly enable the application of different actions based on conditions. It's more of a tool to control loop execution rather than a primary method for implementing conditional logic.

The break statement to immediately exit a loop when a specific condition is met, regardless of whether there are remaining items to iterate over. -> Incorrect. The break statement is used to exit a loop prematurely, which can be useful in certain scenarios. However, it doesn't facilitate the application of different actions based on conditions within the context of iterating over a list of records and performing actions; it simply stops the loop.

Given a requirement to process a list of Account records in Apex and perform different actions based on the Account's AnnualRevenue field, which Apex control flow statement is most appropriate to use?

Correct answer

Use a for loop to iterate over each Account and an if-else statement inside the loop to apply different actions based on the AnnualRevenue field value.

Implement a do-while loop, checking the AnnualRevenue field of each Account as the condition for continuing the loop.

Utilize a switch statement to execute different blocks of code based on the specific value of the AnnualRevenue field for each Account.

Deploy a try-catch block to handle any exceptions that occur based on the AnnualRevenue field values during the iteration process.

Incorrect

Live Agent

Use a for loop to iterate over each Account and an if-else statement inside the loop to apply different actions based on the AnnualRevenue field value. -> Correct. The for loop is ideal for iterating over a collection of records, such as a list of Accounts. The if-else statement within the loop allows for conditional logic to be applied to each Account based on its AnnualRevenue, enabling the execution of different actions depending on the value of this field. This combination effectively addresses the requirement to process and perform actions on a list of records based on specific criteria.

Utilize a switch statement to execute different blocks of code based on the specific value of the AnnualRevenue field for each Account. -> Incorrect. While a switch statement is useful for handling different cases based on the value of a variable, it is less suitable for range-based or more complex conditional logic that might be required for handling numerical fields like AnnualRevenue.

Implement a do-while loop, checking the AnnualRevenue field of each Account as the condition for continuing the loop. -> Incorrect. A do-while loop executes its statements at least once before checking the condition, making it less suitable for conditional processing based on a field value before the loop starts.

Deploy a try-catch block to handle any exceptions that occur based on the AnnualRevenue field values during the iteration process. -> Incorrect. A try-catch block is used for exception handling, not for controlling the flow of program execution based on business logic or field values. It would not be used directly to process records based on the AnnualRevenue field.

A developer needs to update the status of all Case records associated with an Account to “Closed” when the Account status changes to “Inactive”. Which Apex code snippet correctly performs this operation, adhering to Salesforce best practices for bulk data processing?

List casesToUpdate = [SELECT Id FROM Case WHERE AccountId IN (SELECT Id FROM Account WHERE Status = 'Inactive')]; for(Case c : casesToUpdate){ c.Status = 'Closed'; update c; }

Live Agent

Database.executeBatch(new UpdateCaseStatusBatchClass(), 200);

for(Account acc : [SELECT Id FROM Account WHERE Status = 'Inactive']){ List cases = [SELECT Id FROM Case WHERE AccountId = acc.Id]; for(Case c : cases){ c.Status = 'Closed'; update c; } }

Correct answer

List casesToUpdate = new List(); for(Account acc : [SELECT Id FROM Account WHERE Status = 'Inactive']){ casesToUpdate.addAll([SELECT Id FROM Case WHERE AccountId = acc.Id]); } for(Case c : casesToUpdate){ c.Status = 'Closed'; } update casesToUpdate;

Incorrect

```
List casesToUpdate = new List();
for(Account acc : [SELECT Id FROM Account WHERE Status = ‘Inactive’]){
casesToUpdate.addAll([SELECT Id FROM Case WHERE AccountId = acc.Id]);
}
for(Case c : casesToUpdate){
c.Status = ‘Closed’;
}
update casesToUpdate;
```

This code correctly collects Case records in a list and performs a single DML operation outside the loop, adhering to bulk processing best practices.

Incorrect answers:

```
for(Account acc : [SELECT Id FROM Account WHERE Status = ‘Inactive’]){
List cases = [SELECT Id FROM Case WHERE AccountId = acc.Id];
for(Case c : cases){
c.Status = ‘Closed’;
update c;
}
}
```

This code performs DML operations inside a loop, which can hit governor limits in bulk processing scenarios.

Live Agent

Database.executeBatch(new UpdateCaseStatusBatchClass(), 200);

Without seeing the implementation of UpdateCaseStatusBatchClass, it's unclear if this approach adheres to best practices. Additionally, this answer assumes knowledge of batch class implementation details not provided in the question.

```
List casesToUpdate = [SELECT Id FROM Case WHERE AccountId IN (SELECT Id FROM
Account WHERE Status = 'Inactive')];
for(Case c : casesToUpdate){
c.Status = 'Closed';
update c;
}
```

While this code attempts to bulkify SOQL queries, it still performs DML operations inside a loop, risking governor limit breaches.

In a multi-tenant Salesforce environment, a developer is tasked with designing an application that leverages the MVC architecture and the Lightning Component Framework to ensure scalability, maintainability, and tenant isolation. The application must efficiently manage and display customer data within a complex UI. Which of the following approaches best aligns with these requirements?

Creating a single Lightning Component that handles both data retrieval and UI rendering.

Utilizing Visualforce pages for the UI and Apex controllers for business logic, ignoring Lightning Components.

Correct answer

Developing a set of reusable Lightning Web Components, with each component dedicated to a specific part of the UI, and using Apex classes for business logic.

Designing a monolithic Apex application that includes all business logic, with minimal Lightning Components for the UI.

Live Agent

Incorrect

Developing a set of reusable Lightning Web Components, with each component dedicated to a specific part of the UI, and using Apex classes for business logic. -> Correct. This approach adheres to the MVC architecture by clearly separating the model (data), view (UI), and controller (logic) layers. It also takes advantage of the modularity and reusability offered by Lightning Web Components, aligning with multi-tenant concepts by ensuring tenant isolation and efficient resource utilization.

Creating a single Lightning Component that handles both data retrieval and UI rendering. -> Incorrect. This approach violates the separation of concerns principle of the MVC architecture by mixing data management with UI logic, leading to poor maintainability and scalability.

Utilizing Visualforce pages for the UI and Apex controllers for business logic, ignoring Lightning Components. -> Incorrect. While this approach uses MVC principles, it ignores the requirement to leverage the Lightning Component Framework, which is more modern and offers better performance and user experience.

Designing a monolithic Apex application that includes all business logic, with minimal Lightning Components for the UI. -> Incorrect. This approach results in a tightly coupled application that is difficult to maintain and scale. It also does not fully utilize the capabilities of the Lightning Component Framework for creating dynamic and responsive user interfaces.

A developer needs to integrate a custom user interface component into a Salesforce application that enhances user interaction and data processing capabilities. Which approach should the developer take to effectively display and use custom UI components, including Lightning Components, Flow, and Visualforce?

Implement custom UI components using only Apex and Visualforce to ensure maximum compatibility with older Salesforce versions.

Live Agent

Use Salesforce Flow exclusively for all UI components to ensure consistent performance across all devices.

Convert all Visualforce components to Lightning Components by automatically translating Apex code to JavaScript.

Correct answer

Embed a Visualforce page within a Lightning Component using Lightning Out to leverage the strengths of both frameworks.

Incorrect

Embed a Visualforce page within a Lightning Component using Lightning Out to leverage the strengths of both frameworks. -> Correct. This approach allows developers to utilize existing Visualforce pages within the new Lightning Experience, offering a way to integrate complex Visualforce UIs with the modern Lightning Component framework, ensuring a seamless user experience and enhanced capabilities.

Convert all Visualforce components to Lightning Components by automatically translating Apex code to JavaScript. -> Incorrect. While transitioning from Visualforce to Lightning Components is encouraged for leveraging the Lightning platform's capabilities, Apex code cannot be automatically translated to JavaScript. The conversion process requires a manual rewrite of the UI logic appropriate for the Lightning Component framework.

Use Salesforce Flow exclusively for all UI components to ensure consistent performance across all devices. -> Incorrect. While Salesforce Flow is a powerful tool for automating business processes and can be embedded within Lightning pages, it is not suitable as the sole technology for all UI components, especially when custom user interaction or complex UI logic is required.

Implement custom UI components using only Apex and Visualforce to ensure maximum compatibility with older Salesforce versions. -> Incorrect. Relying solely on Apex and Visualforce limits the application's UI capabilities and user experience compared to what can be achieved with Lightning Components. It also does not take full advantage of the modern features and enhancements available in the Lightning platform.

Live Agent

When integrating an external inventory management system with Salesforce, which approach best enables the identification and synchronization of unique inventory items between the two systems, considering the need to create and access the appropriate data model?

Correct answer

Create a custom field on the Salesforce Product object marked as an External ID and populate it with the unique identifier from the external system.

Implement a custom Apex trigger on the Salesforce Product object to generate a new Salesforce ID for each product synced from the external system.

Use a custom formula field on the Salesforce Product object to generate unique identifiers based on product names and categories.

Utilize a standard Salesforce ID field on the Product object as the unique identifier for both the Salesforce and external inventory system.

Incorrect

Create a custom field on the Salesforce Product object marked as an External ID and populate it with the unique identifier from the external system. -> Correct. Utilizing an External ID field is a Salesforce best practice for integrating with external systems. This approach facilitates the efficient upserting (inserting or updating) of records based on a unique identifier that exists outside of Salesforce, ensuring accurate data synchronization between systems.

Use a custom formula field on the Salesforce Product object to generate unique identifiers based on product names and categories. -> Incorrect. A custom formula field cannot serve as a unique identifier for external system integration purposes because it is not designed to handle external IDs and cannot be used to upsert records.

Implement a custom Apex trigger on the Salesforce Product object to generate a new Salesforce ID for each product synced from the external system. -> Incorrect. Generating a new Salesforce ID for each product synced does not address the need to match record between Salesforce and an external system. Salesforce IDs are immutable and automatically assigned by Salesforce upon record creation.

Utilize a standard Salesforce ID field on the Product object as the unique identifier for both the Salesforce and external inventory system. -> Incorrect. Utilizing the standard Salesforce ID field is not practical for synchronizing with an external system because these IDs are unique to Salesforce and have no correlation to external system identifiers.

Live Agent