DUMPSARENA

Certified Force.com Advanced Developer

Salesforce DEV-501

Version Demo

Total Demo Questions: 29

Total Premium Questions: 296

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

QUESTION NO: 1

What three classes along with the Messaging.InboundEmailHandler are used to handle inbound email messages in Salesforce? (No Answer)

A. Apex code, Visualforce pages, and controllers

B. Messaging.InboundEmail, Messaging.InboundEmailResult, Messaging.InboundEnvelope

C. Messaging.InboundEmailHandler

D. Make calls to methods using both valid and invalid inputs.

ANSWER: B

Explanation:

`Messaging.InboundEmail, Messaging.InboundEmailResult, Messaging.InboundEnvelope` is correct because these are the three Messaging namespace types used with the `Messaging.InboundEmailHandler` interface to process an inbound email service request. Salesforce invokes an implementation of the handler interface through its `handleInboundEmail` method. That method receives a `Messaging.InboundEmail` object containing the message content and attachments, plus a `Messaging.InboundEnvelope` object containing envelope-level delivery information, such as the sender and recipient addresses. The implementation then returns a `Messaging.InboundEmailResult`, which communicates the processing result back to Salesforce and can indicate whether the message was successfully handled. Together, these types provide the standard Apex contract for receiving, inspecting, and responding to email sent to an Email Service address. This design allows developers to build server-side automation that creates or updates Salesforce records, routes requests, processes attachments, or performs other business actions based on incoming messages. Salesforce documents the handler method signature and the roles of these parameter and result types in its Messaging.InboundEmailHandler reference and its inbound email Messaging namespace reference.

QUESTION NO: 2

A template component that provides content for an component defined in a Visualforce template page.

See also: ,

A. apex:facet

B. apex:includeScript

C. apex:axis

D. apex:define

ANSWER: D

Explanation:

`apex:define` is the Visualforce template component used by a content page to supply content for a named `apex:insert` placeholder declared in its template. A template page establishes reusable page structure, such as a header, navigation, and footer, and places `apex:insert` tags at locations where individual content pages can contribute their own markup. The content page typically uses `apex:composition` to reference that template, then includes `apex:define` with a `name` attribute matching the corresponding `apex:insert` name. At render time, Visualforce places the body of `apex:define` into that matching insertion point. This pattern separates shared layout from page-specific content, enabling consistent UI structure and simpler maintenance across multiple Visualforce pages. For example, a template can declare `<apex:insert name="body"/>`, while a composed page supplies its primary content through `<apex:define name="body">...</apex:define>`. Salesforce documents this relationship in the Visualforce component references for apex:define and apex:insert.

QUESTION NO: 3

In a recruiting application, all users should be able to see positions with a status of Open. If the status is anything other than Open, the position should be visible only to the record owner.

How would a developer accomplish this? Choose 2 answers

A. Set the organization-wide default for positions to public read-only, then use a sharing rule to restrict access to closed positions.

B. Specify view only access for open positions on users' profiles.

C. Set the organization-wide default for positions to private, then use a criteria-based sharing rule to automatically share open positions.

D. Set the organization-wide default for positions to private, allowing owners to use manual sharing to add or remove access as positions change status.

ANSWER: C

Explanation:

Set the organization-wide default for positions to private, then use a criteria-based sharing rule to automatically share open positions. This configuration implements the required baseline access model: when the Position object's organization-wide default is Private, records are visible to their owners (as well as users with elevated access such as administrators) but are not broadly available to other users. A criteria-based sharing rule can then grant Read Only access when a Position record meets the criterion that its Status equals Open. The rule should share to a public group that contains the intended user population, such as all recruiting application users. Because the rule is criteria-based, Salesforce reevaluates access as the Status value changes. An Open position is shared automatically, and when its Status no longer matches Open, the sharing rule no longer grants that access, leaving the record private to its owner under the organization-wide default. This uses Salesforce's intended additive record-sharing model, combining restrictive defaults with sharing rules that selectively grant access. See Salesforce organization-wide defaults documentation and Salesforce sharing rules documentation.

QUESTION NO: 4

A developer must write a test method for an Apex class that performs callouts to an external recruiting system.

Which approaches can the developer use to test the callout without making a real HTTP request? Choose 2 answers

A. Implement HttpCalloutMock and register it with Test.setMock().

B. Use a static resource callout mock.

C. Set SeeAllData=true so the test can call the external system.

D. Use Test.startTest() without registering a mock response.

ANSWER: A B

Explanation:

The developer can implement the `HttpCalloutMock` interface and register the mock with `Test.setMock()`. The mock returns a controlled HTTP response when the code under test sends its request, allowing the test to verify how the class handles successful, failed, or malformed responses.

The developer can also use a static resource callout mock when response content is stored in a static resource. This approach is useful for reusable JSON or XML response fixtures. Test methods cannot make real callouts, so a mock response is required. Salesforce documents both approaches in Testing HTTP Callouts.

QUESTION NO: 5

A placeholder for content that is rendered in a specific part of the parent component, such as the header or footer of an .

An component can only exist in the body of a parent component if the parent supports facets. The name of the facet component must match one of the pre-defined facet names on the parent component. This name determines where the content of the facet component is rendered. Consequently, the order in which a facet component is defined within the body of a parent component does not affect the appearence of the parent component.

See for an example of facets.

Note: Although you can't represent an directly in Apex, you can specify it on a dynamic component that has the facet. For example:

A. apex:logCallPublisher

B. apex:dataList

C. apex:inputField

D. apex:facet

ANSWER: D

Explanation:

`apex:facet` is correct because it defines named content that a parent Visualforce component renders in a predefined location. A facet is not itself a standalone visual region with a fixed layout; instead, it acts as a placeholder mechanism supported by particular parent components. The facet's `name` attribute identifies the target location exposed by that parent, such as a header or footer area. For example, a data table can use facets to supply specialized markup for a table header or footer while keeping that markup logically associated with the table component. Because placement is determined by the facet name, rather than by the facet's position among the parent component's child tags, developers can organize the markup without changing where the rendered facet appears. This is useful when a component needs structured, reusable presentation regions while allowing custom content in each region. Salesforce's Visualforce component reference documents `apex:facet` as a placeholder for content rendered in a specific portion of its parent component and notes that the parent must support the named facet. See the Salesforce Visualforce reference for apex:facet and the apex:dataTable component reference.

QUESTION NO: 6

To aid _____, each Visualforce page and custom component is saved with version settings for the specified version of the API as well as the specific version of Visualforce.

A. bytecode compiles

B. Apex testing

C. portal development

D. backwards-compatibility

E. exception handling

F. None of these

ANSWER: D

Explanation:

backwards-compatibility is correct. Visualforce pages and custom components store an API version and a Visualforce version so that existing implementations can continue to behave consistently as the Salesforce platform evolves. New platform releases can introduce Visualforce features, adjust framework behavior, or make newer API functionality available. Version settings provide a compatibility boundary: a page authored for an earlier version continues to use the behavior associated with that version unless a developer deliberately updates its version setting and validates the result.

This approach lets Salesforce improve the platform while helping protect previously deployed pages, components, and their controller interactions from unexpected changes. Developers can therefore adopt newer functionality on an intentional schedule rather than having every existing Visualforce artifact automatically take on changed behavior after a release. Salesforce documents that Visualforce pages have a version setting and that the version determines the Visualforce features available to the page. See the Visualforce Developer Guide introduction and Salesforce's Visualforce versioning documentation.

QUESTION NO: 7

Apex code can be initiated in what ways? (Select all that apply)

A. Web service requests.

B. Triggers on objects.

C. Button clicks.

ANSWER: A B C

Explanation:

Apex can run in response to several Salesforce platform entry points. **Web service requests.** is correct because Apex classes exposed with `webservice` methods, or implemented as REST resources, can execute when an external client invokes the published SOAP or REST endpoint. **Triggers on objects.** is correct because an Apex trigger executes automatically before or after specified data events, such as inserting, updating, deleting, or undeleting records. **Button clicks.** is also correct when a user-interface action is configured to invoke Apex, such as a Visualforce page action method, a Lightning component action, or another supported UI mechanism that calls an Apex controller method. In each case, Salesforce starts an Apex transaction in response to the applicable request or record event, then applies the platform's execution context, sharing behavior, and governor limits to that transaction. These are standard ways for declarative record events, external integrations, and user interactions to enter Apex code. See Salesforce's documentation for Apex triggers and calling Apex from Lightning components.

QUESTION NO: 8

What trigger method is used to correlate IDI-to-sObject maps? (No Answer)

A. Trigger.newMap, Trigger.oldMap

B. Internal and external

C. Trigger.new

D. Queues, time triggers

ANSWER: A

Explanation:

`Trigger.newMap, Trigger.oldMap` is correct because these Apex trigger context variables provide direct maps between each record's Salesforce ID and its corresponding sObject record. `Trigger.newMap` is a `Map<Id, sObject>` that contains the new versions of records being processed, while `Trigger.oldMap` contains their prior versions when the trigger operation makes old values available. Developers use the record ID as the shared key to retrieve a specific record efficiently and to correlate its before-and-after state during update or delete-related processing. For example, an update trigger can iterate through the incoming records, use each record's `Id` to obtain the matching old record from `Trigger.oldMap`, and compare field values without performing SOQL queries. This map-based access pattern is bulk-safe because it supports processing all records in the trigger invocation as a collection rather than querying records one at a time. The maps are especially useful for change detection, audit logic, validation, and downstream processing that depends on whether a field value changed. Salesforce documents the trigger context variables and their availability by trigger event in the Apex Trigger Context Variables reference, with additional trigger implementation guidance in the Apex Triggers documentation.

QUESTION NO: 9

Developers can use Visualforce to create a Visualforce page definition. A page definition consists of two primary elements:

Visualforce markup consists of... (Select all that apply)

A. Any Web-enabled code

B. Visualforce Tags

C. Apex code

D. HTML

E. JavaScript

F. C# Remoting Modules

G. None of the above

ANSWER: A B D E

Explanation:

Visualforce markup is the declarative portion of a Visualforce page: it defines what the browser renders and how the page is structured and behaves in the web client. Salesforce describes Visualforce pages as being composed with a combination of Visualforce tags, standard web markup, and client-side web technologies. Therefore, **Visualforce Tags** are valid because they provide Salesforce-specific components and expressions, such as page structure, form controls, output components, and bindings to controller data. **HTML** is valid because standard HTML can be included alongside Visualforce tags to create page layout and browser-rendered content. **JavaScript** is valid because client-side scripts can be used within a Visualforce page to implement browser interactions and integrate supported web behavior. The inclusive wording **Any Web-enabled code** is also valid because Visualforce markup can include other web technologies in addition to the explicitly named examples, subject to Salesforce platform security and page-rendering requirements. Together, these technologies make up the page markup, while server-side controller logic remains a separate element of the overall page definition. See Salesforce's Visualforce Developer Guide introduction and the Trailhead Visualforce page creation unit.

QUESTION NO: 10

A developer is creating a test class for a scheduled Apex class that implements the `Schedulable` interface.

Which steps should the developer use to verify that the scheduled job executes during the test? Choose 2 answers

A. Call System.schedule() with a CRON expression in the test method.

B. Call Test.stopTest() before making assertions about the job results.

C. Wait until the scheduled time occurs in the organization.

D. Invoke the execute method directly instead of scheduling the class.

ANSWER: A B

Explanation:

The test should schedule the job by calling `System.schedule()` with a valid CRON expression and an instance of the schedulable class. This creates the scheduled job in the test context. The test should then call `Test.stopTest()`. Asynchronous work that was enqueued after `Test.startTest()`, including scheduled Apex, runs synchronously when `Test.stopTest()` is called.

The test should make assertions about the results of the scheduled logic after `Test.stopTest()`. It should not rely on a real future scheduled time or invoke the class's `execute` method directly as a substitute for testing scheduling behavior. See Salesforce documentation for Scheduled Apex and Testing Apex.

QUESTION NO: 11

Which of the following guidelines are used for creating custom Web Services? (Select all that apply.)

webservice methods must be static.
webservice methods cannot be overloaded.
A system-defined enum can be used anywhere in a webservice method.
All classes that contain methods defined with the webService keyword must be declared as private.

A. FALSE, they must be static

B. Developer edition production org, Enterprise edition sandbox org

C. Static methods can only be declared in a top-level class definition.

D. webservice methods must be static, webservice methods cannot be overloaded

ANSWER: A D

Explanation:

For an Apex class exposed as a custom SOAP web service, each operation marked with the `webservice` keyword must be a `static` method. Static exposure provides the SOAP runtime with a class-level operation that can be invoked without instantiating the Apex class. In addition, web service methods cannot be overloaded. A generated WSDL must expose operations with unambiguous signatures, so Apex does not support exposing multiple `webservice` methods that share a name but differ by parameters. Therefore, the applicable guidelines are captured by “FALSE, they must be static” (which states the required static behavior) and “webservice methods must be static, webservice methods cannot be overloaded.” In practice, a class intended for external SOAP access is declared `global`, and the exposed methods are declared `global static webservice`. This visibility ensures that the methods can be represented in the generated WSDL and called by external clients. Salesforce’s Apex Developer Guide documents the required declaration form and the restriction against overloaded web-service methods: Apex Web Service Methods. See also the platform guidance for generating a WSDL from an Apex class: Apex Web Services.

QUESTION NO: 12

This integration resource is specific to an individual organization, and exposes all of the standard objects, custom objects, and any custom fields through the SOAP interface.

A. System WSDL

B. Partner WSDL

C. Custom WSDL

D. Enterprise WSDL

E. Portal WSDL

F. None of the above

ANSWER: D

Explanation:

The **Enterprise WSDL** is correct because it is generated specifically for one Salesforce organization. Its schema is strongly typed and reflects the organization’s metadata at the time the WSDL is generated, including the standard objects, custom objects, and custom fields that are available to the generating user. This makes it appropriate when building a SOAP client for a known Salesforce org whose metadata is relatively stable. Generated client code can use explicit types for objects and fields, which supports compile-time validation and a more predictable integration contract. If the organization’s metadata changes—for example, if custom fields or objects are added—the Enterprise WSDL should be regenerated so the integration’s type definitions remain current. Salesforce identifies the Enterprise WSDL as an org-specific WSDL and

describes it as strongly typed, in contrast to WSDLs intended for clients that need to work across multiple organizations. See Salesforce's WSDL generation documentation and the SOAP API WSDL reference for details.

QUESTION NO: 13

A developer needs to provide custom behavior on a Visualforce page that uses the standard controller for the Position custom object. The page must retain the standard controller's existing save and edit functionality.

Which controller type should the developer use?

A. Controller extension

B. Custom controller

C. Apex trigger

D. Standard set controller

ANSWER: A

Explanation:

A controller extension is the correct choice. A controller extension adds custom Apex behavior to a page that already uses a standard controller or another custom controller. The standard controller continues to provide its built-in record operations, such as `save`, `edit`, and `cancel`, while the extension provides additional properties and action methods.

The extension constructor accepts an `ApexPages.StandardController` parameter, enabling the extension to access the current record context. A custom controller would replace the standard controller rather than augment it. Salesforce documents this model in Using Extensions with Standard Controllers.

QUESTION NO: 14

This Apex class allows developers to collect and display errors when building custom controllers or extensions.

A. Exception Class

B. ApexPages.Message Class

C. Debug Class

D. Validation Class

E. Messaging Class

F. None of the above

ANSWER: B

Explanation:

ApexPages.Message Class is correct because it represents an individual page message that a custom controller or controller extension can create for display in a Visualforce page. A developer constructs an `ApexPages.Message` with a severity, such as `ERROR`, and a text value describing the problem. The message is then added to the page's message collection through `ApexPages.addMessage()`. Visualforce components such as `<apex:pageMessages>` render these messages to the user, providing a standard way to show validation failures, processing errors, or other controller-generated feedback without manually managing message markup.

This pattern is particularly useful when an action method needs to retain the user on the current page and communicate one or more actionable issues. Message severity also enables Visualforce to apply the appropriate presentation style for errors, warnings, confirmations, or informational notices. Salesforce documents `ApexPages.Message` as the object used for

custom page messages and `ApexPages.addMessage` as the method that adds one to the current page's message collection. See the ApexPages.Message reference and the ApexPages methods reference.

QUESTION NO: 15

Which action is available to a developer when two objects are connected by a lookup relationship? Choose 2 answers

A. create a roll-up summary field on the parent object to count child records

B. create a custom report type that allows customization of fields displayed from both parent and child objects

C. create a cross-object formula field on the child object to reference fields on the parent object

D. create a cross-object formula field on the parent object to reference fields on the child object

ANSWER: B C

Explanation:

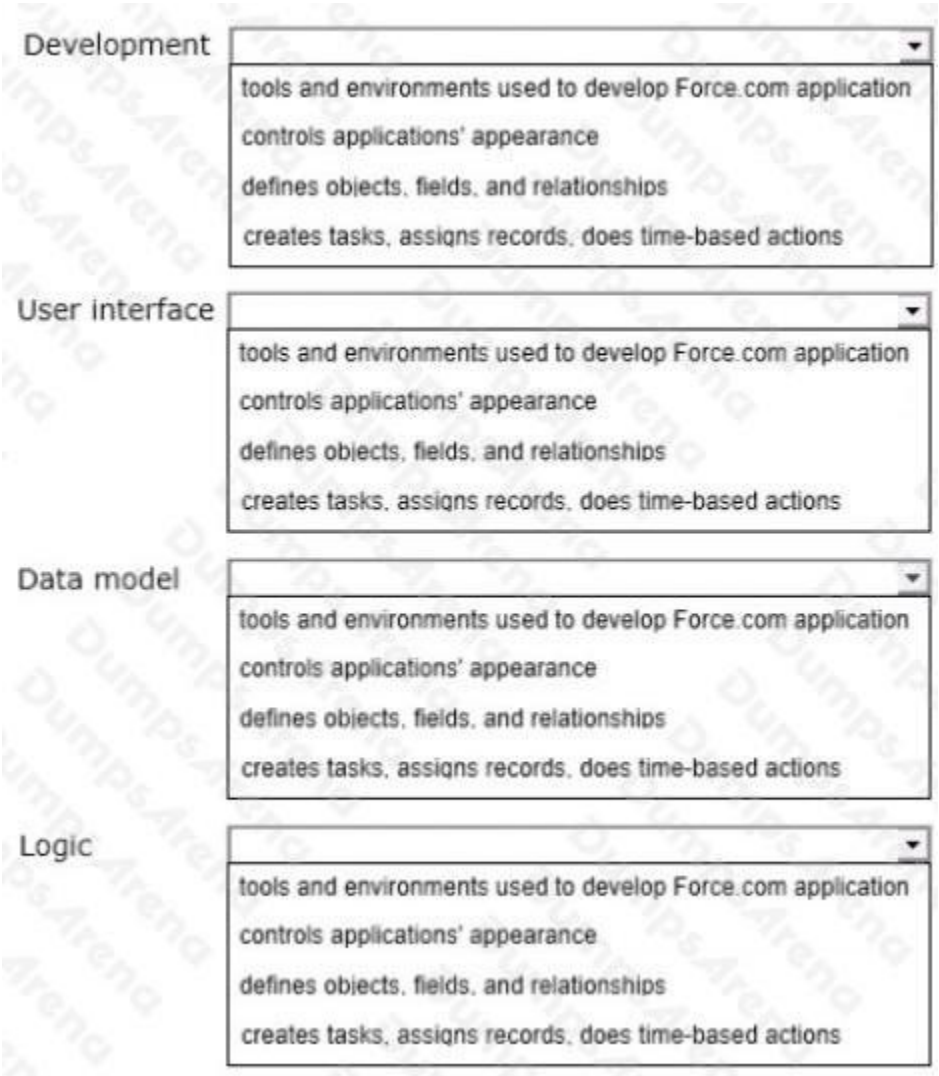
With a lookup relationship, a child record stores a reference to its parent record, so Salesforce can traverse that relationship from the child to retrieve values from the related parent. Therefore, **create a cross-object formula field on the child object to reference fields on the parent object** is available. For example, a formula on a Contact can display a field from its related Account. Cross-object formulas support references through relationship fields and are evaluated dynamically, so they display the current value from the related record without copying data.

It is also possible to **create a custom report type that allows customization of fields displayed from both parent and child objects**. Custom report types define the objects and relationships available to reports, including object relationships based on lookups. A developer or administrator can use the report type layout to select the fields exposed for reporting from both the parent and related child object, enabling reports that combine information across the relationship. Salesforce's relationship documentation describes lookup relationships and their use between related records, while its reporting documentation explains how custom report types define available objects and fields. See Salesforce Object Relationships and Salesforce Custom Report Types Overview.

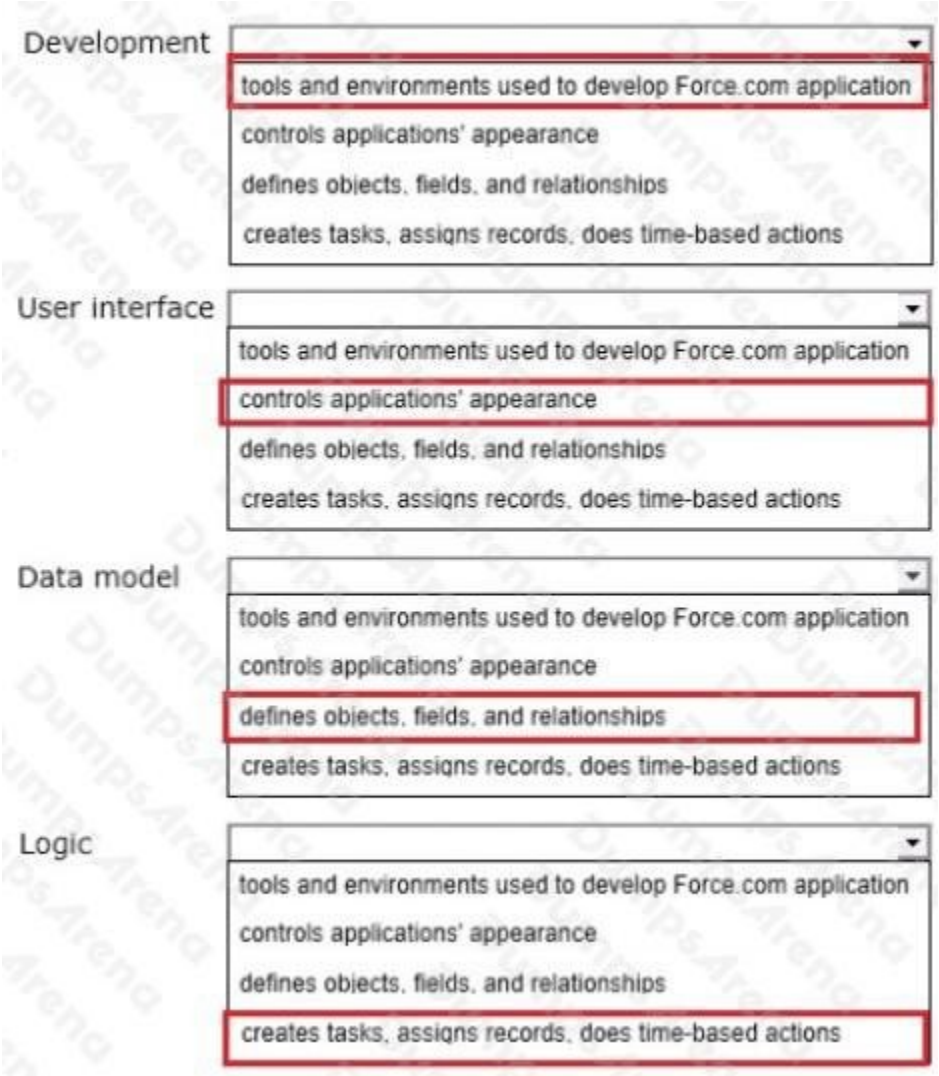
QUESTION NO: 16 - (HOTSPOT)

Match the Force.com platform aspects with their corresponding descriptions.

Hot Area:


Development
tools and environments used to develop Force.com application
controls applications' appearance
defines objects, fields, and relationships
creates tasks, assigns records, does time-based actions
User interface
tools and environments used to develop Force.com application
controls applications' appearance
defines objects, fields, and relationships
creates tasks, assigns records, does time-based actions
Data model
tools and environments used to develop Force.com application
controls applications' appearance
defines objects, fields, and relationships
creates tasks, assigns records, does time-based actions
Logic
tools and environments used to develop Force.com application
controls applications' appearance
defines objects, fields, and relationships
creates tasks, assigns records, does time-based actions

ANSWER:



Explanation:

The correct mapping reflects the way Salesforce separates application construction into platform layers. Development refers to the tools and environments used to build Force.com applications. These include the Salesforce development capabilities used to create, test, deploy, and maintain custom functionality, such as Apex, Visualforce, declarative tooling, and development environments. Salesforce's platform overview describes the platform capabilities available for building applications.

User interface controls an application's appearance because this layer is responsible for what users see and interact with. Page layouts, Lightning pages, components, styling, navigation, and other presentation elements determine how information and actions are displayed to users. The interface layer presents the application's data and behavior in a usable form.

Data model defines objects, fields, and relationships. Objects provide the structured records used by an application, fields store attributes on those records, and relationships connect records across objects. This structure is the foundation for storing, retrieving, and organizing business data. Salesforce documents these concepts in its data modeling guidance.

Logic creates tasks, assigns records, and performs time-based actions because automation and business-process logic determine what should happen when records meet defined conditions or when scheduled criteria are reached. Salesforce automation can update records, create follow-up work, route ownership, and execute actions based on time. The selected descriptions therefore correctly distinguish development tooling, presentation, data structure, and automated business behavior.

QUESTION NO: 17

A developer has added a custom object tab to an application. Which additional feature will become available by default for the object in the application? Choose 3 answers

A. Create New sidebar component

B. Custom reporting

C. Recent items

D. Quick create

E. Search

ANSWER: C D E

Explanation:

Adding a custom object tab to an application exposes the object as a first-class, navigable part of that application. The object's records can therefore participate in the application's recent-item behavior: when a user opens a record through the tab, it can be surfaced in **Recent items**. The tab also provides a normal object entry point for creating records, making **Quick create** available to users who have the required object-level create permission. In addition, records for the tabbed custom object are available to the platform's search experience, so **Search** can return matching records when the user has permission to access them. These capabilities make the custom object usable in the app beyond simply displaying its tab, while Salesforce sharing, CRUD permissions, and field-level security continue to govern what individual users can find, open, or create. Salesforce's custom-object model and object metadata are described in the CustomObject reference; see also Trailhead's overview of objects and custom objects.

QUESTION NO: 18

Which developer tool can be used to create a data model? Choose 2 answers

A. Force.com IDE

B. Schema Builder

C. Application Data Model Wizard

D. Force.com Data Loader

ANSWER: A B

Explanation:

Schema Builder is designed specifically for visually creating and maintaining a Salesforce data model. It provides a diagram of objects and their relationships, and allows a developer to create custom objects, custom fields, and relationship fields directly from the canvas. This makes it useful for modeling how business data is structured and connected before or while building application logic. Salesforce describes Schema Builder and its supported metadata operations in the Schema Builder documentation.

Force.com IDE can also be used to create a data model in the context of this exam. It is an Eclipse-based development tool that works with Salesforce metadata, including custom-object definitions and field metadata. A developer can retrieve metadata into a project, create or modify object-definition files, and deploy the resulting metadata to an org. Because custom objects, fields, and relationships define the data model, editing and deploying that metadata constitutes creating or changing the model. The underlying metadata structure for custom objects is documented in the Salesforce Metadata API CustomObject reference.

QUESTION NO: 19

A developer needs to support multiple currencies for a custom object in an application. The multi-currency feature has been enabled for the organization.

What does the developer need to know in order to successfully support this application? Choose 2 answers

A. Administrators can add additional currencies after the application has been built.

B. Roll-up summary fields on a parent record will calculate incorrectly if the child records have multiple currencies.

C. The multi-currency feature will automatically track historical exchange rates.

D. Currency ISO code will need to be supplied when inserting records in different currencies.

ANSWER: A D

Explanation:

Administrators can add additional currencies after the application has been built. is correct because an organization's active currency list is an administrative configuration, rather than a fixed characteristic of a custom object or its Apex code. Once multiple currencies are enabled, administrators can activate and manage additional currencies as business requirements expand. Applications should therefore be designed to work with currency values dynamically instead of assuming a permanently limited set of currencies.

Currency ISO code will need to be supplied when inserting records in different currencies. is also correct. Currency-enabled records include the standard `CurrencyIsoCode` field. Apex, API integrations, and data-loading processes that create a record in a currency other than the applicable default must set this field to the appropriate active ISO currency code, such as `EUR` or `JPY`. This ensures the monetary fields on the custom-object record are stored and displayed in the intended currency. If no currency is supplied, Salesforce uses the applicable default currency context, which does not reliably meet a requirement to create records in distinct currencies.

See Salesforce's guidance on managing currencies and the API documentation for the standard CurrencyIsoCode field type.

QUESTION NO: 20

What language is Apex similar to? (No Answer)

A. Catch

B. 1

C. TRUE

D. Java

ANSWER: D

Explanation:

Java is correct. Apex is Salesforce's strongly typed, object-oriented programming language, and its syntax is deliberately modeled after Java. Developers familiar with Java will recognize core language constructs such as classes, interfaces, methods, variables, conditional statements, loops, exception handling, collections, and object-oriented design patterns. Apex also uses Java-like punctuation and block syntax, including semicolons and curly braces, which makes its code structure readily familiar to Java developers. Salesforce describes Apex as a strongly typed, object-oriented language with syntax that looks like Java and acts like database stored procedures in the Salesforce platform. Although Apex is purpose-built for the multitenant Salesforce environment and includes platform-specific features such as SOQL, SOSL, DML operations, triggers, governor limits, and built-in access to Salesforce records, its closest language comparison remains Java. This similarity helps developers transfer foundational Java knowledge when writing Apex classes, triggers, asynchronous jobs, and test methods. See Salesforce's Apex Developer Guide introduction and the Trailhead introduction to Apex for the language's syntax and platform context.

QUESTION NO: 21

Good Apex tests should....

A. Exercise bulk trigger functionality—use at least 20 records in your tests.

B. Only test code using the Standard User profile.

C. Access live organization data to validate SOQL queries and DML operations.

D. Use the isTest annotation. Classes defined with the isTest annotation do not count against your organization limit of 3 MB for all Apex code. See IsTest Annotation.

E. Use the runAs method to test your application in different user contexts.

F. None of these

ANSWER: A D E

Explanation:

Good Apex tests should exercise code under realistic platform conditions while remaining independent, repeatable, and easy to maintain. “Exercise bulk trigger functionality—use at least 20 records in your tests” is correct because Apex triggers and other database-processing code must handle collections of records rather than relying on single-record behavior. Creating a meaningful set of test data helps validate that logic is bulkified and operates within governor limits. Test code should also use the `@isTest` annotation. This identifies a class as test-only code and excludes it from the organization’s Apex code size limit, allowing test suites to support production code without consuming that allocation. Finally, `System.runAs` is an important testing tool for code whose behavior depends on the running user. It lets a test execute a block in another user context, supporting validation of sharing-sensitive behavior and user-specific execution paths. Tests still run with test-created data by default, so they can reliably establish the users, records, and conditions required for each scenario. Salesforce’s guidance on writing and running tests is available in the Apex Testing documentation; details on test annotations are in the @isTest Annotation documentation.

QUESTION NO: 22

A developer has written a custom Apex controller declared `with sharing` for a Visualforce page that displays Position records.

Which statements are TRUE? Choose 2 answers

A. The controller enforces the current user's record-level sharing rules.

B. The controller does not automatically enforce object permissions and field-level security.

C. The controller always runs in user mode for all SOQL and DML operations.

D. The controller bypasses sharing rules because Visualforce pages run in system context.

ANSWER: A B

Explanation:

A class declared `with sharing` enforces the current user's record-level sharing rules for SOQL queries and DML operations performed by that class. Therefore, the controller does not automatically return Position records that the running user is not permitted to access through the sharing model.

The `with sharing` declaration does not enforce object-level permissions or field-level security. Apex generally runs in system mode for CRUD and FLS unless the developer explicitly enforces those permissions, for example by using user-mode database operations, `WITH USER_MODE`, or appropriate security checks. Salesforce documents the distinction in Using the with sharing, without sharing, and inherited sharing Keywords and Secure Server-Side Development.

QUESTION NO: 23

An HTML input element of type hidden, that is, an input element that is invisible to the user. Use this component to pass variables from page to page.

A. apex:inputHidden

B. apex:actionPoller

C. apex:pageBlockSectionItem

D. apex:enhancedList

ANSWER: A

Explanation:

`apex:inputHidden` is correct because it renders an HTML `<input>` element whose type is `hidden`. The field is included in the submitted form data but is not displayed in the page UI, allowing a Visualforce page to retain or submit a value associated with a controller property. Its `value` attribute can be bound to an Apex controller or extension variable using a Visualforce expression, such as `{!someProperty}`. When the Visualforce form is submitted through a command component or action function, the bound hidden-field value participates in the request and can be processed by the controller. This makes the component useful when a page needs to carry an identifier, state value, or other non-user-editable form value during a request. As with all client-side hidden inputs, developers should not treat the browser-provided value as trusted: authorization, CRUD/FLS enforcement, and server-side validation remain necessary before using a submitted value. Salesforce documents this component as a Visualforce input component that generates a hidden HTML input field and supports value binding. See the apex:inputHidden component reference and the Visualforce expressions and value binding documentation.

QUESTION NO: 24

Object X has a lookup field to Object Y. X needs to display a text value from a Text field on Y. To ensure data Integrity, how would a developer implement this?

A. Create a roll-up summary field on Object X that retrieves the value from Y.

B. Create a text field on Object X and use a workflow rule to fill in the value upon the creation of X

C. Create a cross-object formula field on Object X that retrieves the value from Y.

D. Create a text field on Object X and use Apex to populate the value.

ANSWER: C

Explanation:

Create a cross-object formula field on Object X that retrieves the value from Y. A cross-object formula can traverse Object X's lookup relationship to reference the text field on the related Object Y record. Formula fields are calculated dynamically whenever the record is accessed, rather than storing a copied value on Object X. Consequently, if the text value on Object Y changes, the value displayed on Object X immediately reflects that change without requiring automation, Apex, or a separate update process. This supports data integrity by maintaining Object Y as the single source of truth and avoiding duplicated data that could become stale or inconsistent. The formula field should use the relationship name in its expression, such as `Object_Y__r.Text_Field__c`, with the exact API names depending on the org's configuration. Salesforce supports cross-object formulas for accessing fields through lookup and master-detail relationships, and formula fields do not consume database storage because their values are derived at runtime. See Salesforce's Cross-Object Formula Fields documentation and the Custom Field Types documentation for details.

QUESTION NO: 25

What is the size of the batches in which triggers execute?

A. No

B. 200

C. FALSE

D. Java

ANSWER: B

Explanation:

200 is correct. Apex triggers execute on collections of records rather than one record at a time, and the maximum trigger batch size is 200 records. For example, when a bulk API operation, data import, or user action processes many records, Salesforce divides the operation into batches of up to 200 records and invokes the relevant trigger once for each batch. The trigger context collections, such as `Trigger.new`, `Trigger.old`, `Trigger.newMap`, and `Trigger.oldMap`, can therefore contain as many as 200 records in a single invocation. Developers should write bulkified trigger logic that iterates over these collections and performs SOQL queries and DML operations on sets or lists outside loops. This design ensures that code works correctly both for a single-record save and for a full batch, while staying within the per-transaction governor limits that apply to the trigger execution. Salesforce documents 200 as the maximum batch size for Apex triggers and identifies the trigger context variables used to access all records in the current batch. See Apex Trigger Context Variables and Apex Governor Limits.

QUESTION NO: 26

A developer can use optional catch statements for any exception type in a try-catch block. However, the general exception type, 'Exception', must only be used by the last catch() block.

EXAMPLE:

try{

// Some risky code.

}

catch(SomeExceptionType e){

// Handle one exception type.

}

catch(SomeOtherExceptionType e){

// Handle another exception type.

}

catch(Exception e){

// This must be the last catch block.

}

~|~

(Select all that apply)

A. Make calls to methods using both valid and invalid inputs.

B. In the case of conditional logic (including ternary operators), execute each branch of code logic.

C. Only test code using the System Administrator profile.

D. Focus solely on test coverage percentage score.

E. Complete successfully without throwing any exceptions, unless those errors are expected and caught in a try...catch block.

F. None of these

ANSWER: A B E

Explanation:

Effective Apex tests validate both successful behavior and expected failure behavior rather than exercising only a happy path. “Make calls to methods using both valid and invalid inputs.” is correct because tests should confirm that business logic handles valid data correctly and responds predictably when supplied invalid, incomplete, or boundary-case data. This helps verify validation, error-handling paths, and the application’s intended contract.

“In the case of conditional logic (including ternary operators), execute each branch of code logic.” is also correct. A meaningful test suite drives each possible branch of a decision so that the outcomes of the business rules are validated, not merely counted as covered. Salesforce requires sufficient code coverage for deployment, but recommends tests that assert expected results and cover the relevant paths through application logic.

“Complete successfully without throwing any exceptions, unless those errors are expected and caught in a try...catch block.” is correct because test methods must finish without unhandled exceptions. When a failure is an intended outcome, the test can deliberately invoke it and use exception handling plus assertions to verify the expected error behavior. Salesforce documents these testing practices in its Apex testing best practices and Apex testing documentation.

QUESTION NO: 27

If the error message contains HTML markup, the escaped markup displays as text and isn't rendered in the user interface. (Select all that apply)

A. Strongly Typed.

B. Object Oriented.

C. Uses Java-like syntax.

D. Acts like database stored procedures.

ANSWER: A B C D

Explanation:

The listed characteristics accurately describe Apex. Apex is a strongly typed language: variables, expressions, and method calls have defined data types that are checked during compilation. It is also object-oriented, supporting classes, interfaces, inheritance, encapsulation, and polymorphism, which lets developers organize reusable business logic in familiar application structures. Apex uses Java-like syntax, including braces, control-flow constructs, classes, and method definitions, while remaining purpose-built for the Salesforce platform. Finally, Apex code executes on Salesforce servers in a way comparable to database stored procedures: it can query and manipulate Salesforce data, enforce transactional behavior, and run close to the platform’s data layer. Salesforce documentation describes Apex as a strongly typed, object-oriented programming language with Java-like syntax, and explains that it enables developers to add business logic to system events and execute database operations. See the Salesforce Apex overview and the Apex Developer Guide introduction.

QUESTION NO: 28

Which keywords do developers use to handle exceptions in Apex?

A. Through class itself

B. Throw, try, catch, finally

C. Static and final

D. GET, POST, PUT, DELETE

ANSWER: B

Explanation:

Throw, try, catch, finally is correct because Apex provides these language keywords for structured exception handling. Code that could produce an exception is placed in a `try` block. A corresponding `catch` block receives and handles an exception when one is thrown, allowing the application to respond in a controlled way, such as logging an error, showing an appropriate message, or performing recovery logic. Developers use `throw` to explicitly raise an exception, including a custom exception, when application logic detects an invalid or unsupported condition. The optional `finally` block runs after the try-and-catch processing completes, whether an exception occurred or not, so it is useful for cleanup operations that must always occur. Apex exception handling follows this familiar try-catch-finally model while also supporting built-in exception types and developer-defined exception classes. Proper handling helps make Apex transactions more predictable and enables code to communicate meaningful failures rather than allowing unhandled errors to terminate processing without application-specific handling. Salesforce documents the syntax and behavior of these keywords in its Apex Exception Handling documentation and the Exception Class and Built-In Exception Methods reference.

QUESTION NO: 29 - (SIMULATION)

Match the Force.com platform aspects with their corresponding descriptions.

ANSWER: Check the explanation for the answer

Explanation:

The simulation payload does not include the draggable platform aspects or the description choices, so an exact one-to-one match cannot be determined from the supplied text alone.

The standard Force.com platform-aspect matches commonly tested are:

Multitenant architecture — Multiple organizations share the same underlying infrastructure while their data and configuration remain isolated.
Metadata-driven development — Applications are defined through configuration metadata (objects, fields, layouts, workflows, and so on) rather than requiring infrastructure code.
Model-View-Controller (MVC) — The data model is represented by objects; the view is the user interface/Visualforce page; the controller contains application logic.
Declarative development — Administrators can create much of an application using point-and-click tools, without writing code.
Apex — Salesforce's server-side, strongly typed programming language for custom business logic, triggers, and controllers.
Visualforce — The component-based markup framework used to build custom user interfaces and pages.
API/integration services — Standard web-service APIs used to access Salesforce data and integrate external applications.

Use these mappings against the exact labels shown in the simulation. The missing answer options are needed to identify the precise expected set.