PassQueen
a.k.a dumpsqueen.com

Magento Front End Developer Certification Exam

Magento M70-301

Version Demo

Total Demo Questions: 14

Total Premium Questions: 143

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Working with themes	37
Topic 2, Working with layouts	76
Topic 3, Working with JavaScript	12
Topic 4, Working with CSS	11
Topic 5, Mix Questions	7
Total	143

QUESTION NO: 1

Which two of the following statements are true regarding Magento configuration scopes? (Choose TWO.)

A. “Websites” contain “Stores”.

B. A “Store” can be associated with multiple “Websites”.

C. A “Store View” can be associated with multiple “Stores”.

D. A “Store” can be associated with multiple “Store Views”.

E. Multiple “Websites” can share a “Store View”.

ANSWER: A D

Explanation:

Magento uses a hierarchical store structure in which websites are the top organizational level, stores belong to a website, and store views belong to a store. Therefore, “Websites” contain “Stores” is true: a website may have one or more stores, and a store is assigned to one website. This hierarchy supports operating multiple sites with separate customer, catalog, and pricing contexts where appropriate.

A “Store” can be associated with multiple “Store Views” is also true. Store views are commonly used for different languages, locale-specific presentation, or other storefront-display variations while sharing the same underlying store. Configuration values can be defined at default, website, or store-view scope, with more specific scoped values overriding values inherited from broader levels when a setting supports that scope. This hierarchy is important when configuring catalog behavior, localization, design, and other storefront settings in the Magento Admin.

Adobe’s documentation describes the website, store, and store-view hierarchy and its role in managing multistore installations. See Stores, websites, and store views and Changing configuration scope.

QUESTION NO: 2

In which of the following directories is a .xml file located?

A. CSS

B. Layout

C. Skin

D. Template

ANSWER: B

Explanation:

Layout is correct. In the Magento theme structure relevant to the Front End Certified Developer examination, layout update instructions are stored as XML files in the theme’s `layout` directory, such as `app/design/frontend/<Vendor>/<theme>/layout`. These files define the page structure and presentation instructions that Magento applies when rendering requests. A layout XML file can declare containers and blocks, assign templates to blocks, add or remove page elements, set block arguments, and control the order in which content is rendered. Magento merges applicable layout updates from modules and the active theme to build the final page layout for a handle, such as a catalog product page or checkout page. This separation lets a frontend developer modify page composition without changing PHP application logic. Adobe’s layout documentation describes layout XML as the mechanism used to define and arrange page elements, including containers and blocks, and explains how layout updates are processed for a requested page. See the Adobe Commerce Layouts guide and the Layout XML instructions reference.

QUESTION NO: 3

You want to add a block to the content of the product detail page, without having to modify a template.

Which three container blocks are rendered regardless of product type and configuration? (Choose THRE

E.
)

A. alert.urls

B. product.info.extrahint

C. product.info.options.wrapper

D. product.info.options .wrapper.bottom

E. product.info.additional

F. product.info.container1

G. product.info.container2

ANSWER: A B E

Explanation:

`alert.urls`, `product.info.extrahint`, and `product.info.additional` are suitable insertion points because the standard Magento product-view template renders their child HTML as part of the general product detail-page structure. These blocks are not limited to a particular product-type renderer, such as the simple, configurable, grouped, bundle, or downloadable product blocks. Consequently, a layout update can reference one of these blocks and add a child block or CMS static block without editing the product-view template. `alert.urls` is rendered in the product-shop area and is intended for product alert links. `product.info.extrahint` is an extension point rendered with the core product information. `product.info.additional` is rendered in the product collateral area, allowing additional content to be placed alongside product-detail sections. Magento's layout system is specifically designed to arrange blocks and inject child blocks through XML layout updates; child output is included when the parent template calls `getChildHtml()`. The relevant product-view layout and template definitions can be reviewed in the Magento 1 source: catalog.xml product-view layout and product view template.

QUESTION NO: 4

How can you minimize the number of HTTP requests made by your native Magento sites for JavaScript files?

A. Enable JavaScript merging in the admin.

B. Move JavaScript requests to footer block in local.xml.

C. Enable all Cache Storage Management.

D. Remove Magento JavaScript and only use files hosted by a third party CDN.

ANSWER: A

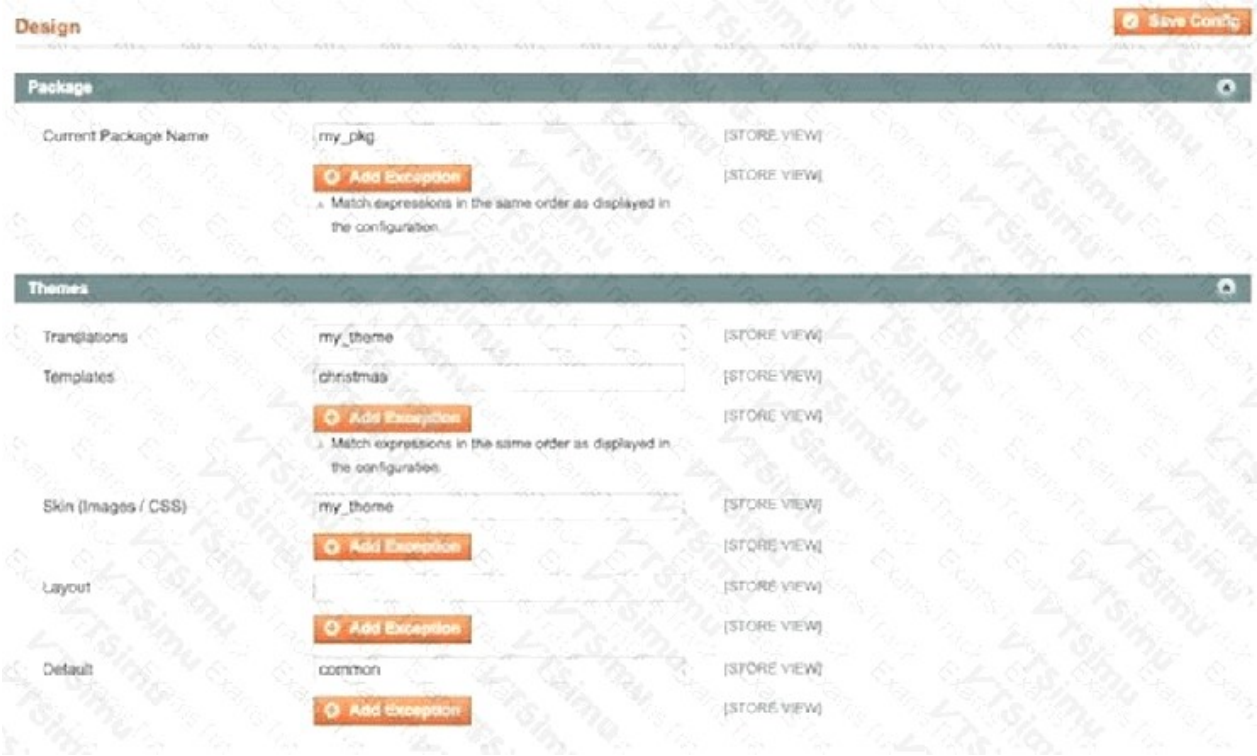
Explanation:

Enable JavaScript merging in the admin. is correct because Magento's JavaScript merge feature combines multiple JavaScript source files into a smaller number of generated, merged files. Since a browser ordinarily makes a separate HTTP request for each JavaScript asset referenced by a page, reducing those separate assets directly reduces the number of JavaScript-related HTTP requests. In Magento 1, this setting is available in the Admin configuration under the Developer settings, in the JavaScript Settings section. Once enabled, Magento builds merged JavaScript output in its media directory and page layouts reference the merged resource rather than the individual files where merging is applicable.

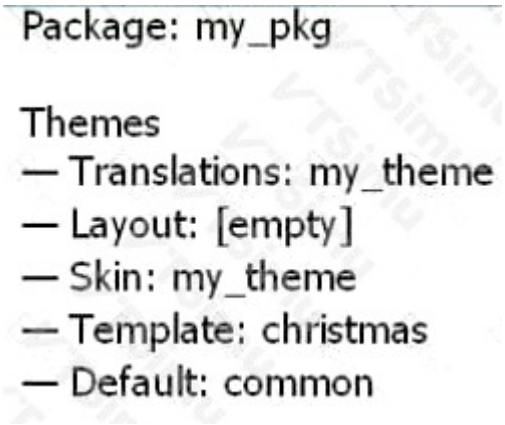
This is a front-end performance optimization intended to reduce request overhead, which was particularly important with the connection limits and request costs common to the browser and HTTP/1.1 environments targeted by Magento 1. The feature

should be enabled and tested in an environment representative of production, because custom scripts must still preserve their required load order and dependencies. Magento's page configuration defines the merge setting in its core system configuration: Magento Page module configuration. For current Adobe Commerce guidance on deploying and optimizing static assets, see Static view file deployment documentation.

QUESTION NO: 5



Magento is searching in the fallback system for a my-template.phtml file. As shown in the graphic above, you have configured Magento to the following:



In which four places might Magento find that file, assuming it has been placed there? (Choose FOUR)

A. app/design/frontend/my_pkg/my_theme/template/

B. app/design/frontend/default/default/

C. app/design/frontend/base/default/layout/

D. app/design/frontend/my_pkg/common/template

E. app/design/frontend/my_pkg/christmas/template

F. skin/frontend/my_pkg/christmas/template

G. skin/frontend/my_pkg/common/

ANSWER: D E

Explanation:

`app/design/frontend/my_pkg/christmas/template` is a valid location because Magento resolves a block template through the configured frontend design package and the active template theme. A template requested as `my-template.phtml` is therefore sought beneath that theme's `template` directory, using its relative template path. `app/design/frontend/my_pkg/common/template` is also valid because Magento's design fallback mechanism can continue to the configured fallback/parent theme within the same package when the file is not present in the active theme. This permits a package to keep shared PHTML files in a common theme while a specialized theme such as `christmas` provides only its overrides. The fallback lookup concerns design-area template directories, so the package and theme names must be resolved in the frontend design context and the file must reside under the `template` directory. Magento's core design-package implementation performs this theme fallback resolution before returning the template file path. See the Magento/OpenMage design package source and Adobe's theme structure documentation.

QUESTION NO: 6

Given the information shown below, which answer will correctly assign a customized template file using layout XML?

Block type:

example/controller

Template path:

/a pp/design/frontend/base/exampletheme/examplefolder/example.phtml

A.

B.

C.

D.

ANSWER: C

Explanation:

`<block type="example/controller" name="example" as="example" template="examplefolder/example.phtml" />` is correct because Magento 1 layout XML creates a block with the `block` element and identifies its PHP block class through the `type` attribute. The supplied block type, `example/controller`, therefore belongs in that attribute. The `template` attribute assigns the PHTML file to the block. Its value is relative to the active design package/theme's `template` directory, rather than an absolute filesystem path. Consequently, `examplefolder/example.phtml` resolves to `app/design/frontend/base/exampletheme/template/examplefolder/example.phtml` when that package and theme are active. The `name` attribute gives the layout block a unique identifier, while `as` provides the alias by which a parent block can render or retrieve it. This follows Magento's layout structure: layout XML instantiates blocks, configures block attributes, and associates template files with template-capable blocks. See the Magento 1-compatible layout XML documentation and the theme and template documentation.

QUESTION NO: 7

Which element in layout XML specifies a method to be called on a referenced or newly defined block?

A. < action >

B. < call >

C. < change >

D. < method >

E. < update >

ANSWER: A

Explanation:

The `<action>` element is used in Magento layout XML to invoke a public method on a block that has just been declared or that is being referenced elsewhere in the layout. Its `method` attribute identifies the block method to execute, and any nested XML elements supply that method's arguments. For example, an action such as `<action method="setTemplate"><template>page/1column.phtml</template></action>` calls `setTemplate()` on the target block and passes the template value to it. This mechanism lets layout updates configure block behavior declaratively, without requiring PHP changes for each layout handle. In Magento's layout processing implementation, action nodes are collected and interpreted as method calls against the relevant block instance, which is why the element represents the invocation itself while the method name is an attribute. The Magento codebase's layout model and its standard layout XML files illustrate this established layout instruction pattern; see the Magento-compatible layout model implementation and the base frontend layout XML files.

QUESTION NO: 8

You want to override the lcolumn.phtml file. Which two of the following methods could you use? (Choose TWO.)

A. Rename the file in base/default to enable overriding.

B. Rename the root template configuration in the admin.

C. Create a file with same file name in the same directory structure as your theme.

D. Create a file exception in System -> Design, clear cache, and rebuild indexes.

E. Create a file with different file name and different directory structure; then change the template with the setTemplate method.

ANSWER: C E

Explanation:

Magento 1 resolves template files through the active design package and theme fallback hierarchy. Therefore, creating a file named `lcolumn.phtml` in the corresponding directory under the custom theme is a valid override: when Magento requests that template path, it finds the custom-theme version before falling back to the base theme's copy. This is the standard approach when the replacement should transparently apply anywhere the original template path is used.

A template can also be overridden by supplying a different template file and changing the relevant block's template value with `setTemplate()`. Layout XML is commonly used to make that call for a named block, allowing the replacement template to have its own filename and location. Magento will then render the newly assigned template instead of the original `lcolumn.phtml` file. This approach is useful when the override is intended for a specific layout handle, page type, or block instance rather than every use of the original template path. Magento's template and layout mechanisms are documented in the Magento 1 layout documentation and the Magento 1 design and template documentation.

QUESTION NO: 9

Which three of the following action method(s) are valid ways to add custom JavaScript files? (Choose THREE.)

A.

B. jsexample.js

C. skin_jsexample.js

D. skin_jsexample.js

E. js

F.

G. jsexample.js

ANSWER: A D G

Explanation:

Magento 1 layout XML can add JavaScript assets through methods exposed by the HTML head block. The corrected `<action method="addJs"><script>example.js</script></action>` calls `addJs()`, which accepts the JavaScript file path and registers it as a standard JavaScript asset for rendering in the page head. This is the concise layout-XML form for a file served from the configured JavaScript directory.

The corrected `addItem` entries use the head block's more general asset-registration method. With `<type>skin_js</type>`, Magento treats the named file as a skin JavaScript resource, resolving it through the active design package and theme. With `<type>js</type>`, Magento registers a normal JavaScript resource. In both cases, the `name` element supplies the file name or relative path. These methods support Magento's asset collection and head rendering behavior, rather than directly emitting a script tag. The Magento 1 head-block implementation defines `addJs()` and `addItem()` and shows the supported item types: Mage_Page_Block_Html_Head source. Magento's layout XML concepts are also documented in the Magento 1 Developer Guide.

QUESTION NO: 10

Given the layout XML example below, which one of the following actions will cause the title, "Some Title," to be translated by Magento?

EXAMPLE:

```
<action method="something">
    <title>Some Title</title>
    <message>Some message</message>
</action>
```

A. Add the strings ("Some Title","Ein Titel") to a translation CSV file.

B. Change the XML as follows:

```
<action method="something" translate="title">
    <title>Some Title</title>
    <message>Some message</message>
</action>
```

C. Change the XML as follows:

```
<action method="something">
    <title translate="en_US, de_DE">Some Title</title>
    <message>Some message</message>
</action>
```

D. Change the XML as follows:

```
<action method="something">
    <title><en_US>Some Title</en_US><de_DE>Some Title</de_DE></title>
    <message>Some message</message>
</action>
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: B

Explanation:

Magento translates values supplied through layout XML actions when the action explicitly identifies the applicable argument as translatable. For a page-title action, this is done by adding a `translate` attribute that names the child argument containing the title, such as `translate="title"` on an action with a `<title>Some Title</title>` child node. Magento's layout processing then passes that value through the translation system before invoking the target block method. Where required, the module context supplied on the action determines the translation dictionary in which Magento looks for the matching phrase. Consequently, the title can be localized according to the active store locale rather than always rendering the literal English phrase from the layout file. This behavior is important because layout XML values are not automatically translated merely because they are text; the XML must declare which action argument Magento should translate. Adobe's translation guidance describes the locale-based phrase translation mechanism and the use of module translation dictionaries: Adobe Commerce translations guide. The Magento layout XML model and the relationship between layout instructions, blocks, and actions are also documented in the Adobe Commerce layout guide.

QUESTION NO: 11

Which two of the following would add a CSS file to every page? (Choose TWO.)

A.
```
<default>
    <reference name="head">
        <action method="addCss">css/styles.css</action>
    </reference>
</default>
```

B.
```
<default>
    <reference name="head">
        <action method="addItem">
            <type>skin_css</type>
            <name>css/styles.css</name>
        </action>
    </reference>
</default>
```

C.
```
<default>
    <reference name="head">
        <action method="addStylesheet">css/styles.css</action>
    </reference>
</default>
```

D.
```
<default>
    <reference name="head">
        <action method="addCss">
            <stylesheet>css/styles.css</stylesheet></action>
    </reference>
</default>
```

E.
```
<default>
    <reference name="head">
        <action method="addItem">
            <type>css</type>
            <name>css/styles.css</name>
        </action>
    </reference>
</default>
```

A. Option A

B. Option B

C. Option C

D. Option D

E. Option E

ANSWER: B D

Explanation:

The two marked selections are correct because Magento storefront CSS assets can be declared in the `head` section of a layout XML update that is applied globally. Magento merges layout XML files from modules and the active theme to build each page's layout; a CSS instruction in a global/default layout handle therefore produces a stylesheet link on every storefront page that uses that handle. In a theme, the conventional location for global head assets is the `Magento_Theme` layout area, particularly a global head-blocks layout update. A CSS asset may also be added through the global default page layout update, provided that the declaration is nested in the page `head` element. In both cases, the stylesheet source is resolved as a theme static asset and deployed or generated according to the current application mode. This mechanism is preferable to manually placing a `<link>` tag in a template because layout XML preserves Magento's asset-resolution,

merging, and static-content deployment behavior. Adobe's layout documentation describes layout XML instructions and the page head container at Adobe Commerce layout overview. Theme stylesheet customization and static asset handling are covered in the Adobe Commerce CSS customization guide.

QUESTION NO: 12

Which code snippet shows the correct way to add a JavaScript file from your custom theme to all Magento pages?

A. Option A

B. Option B

C. Option C

D. Option D

E.

ANSWER: E

Explanation:

To include a theme-owned JavaScript asset on every storefront page, declare the asset in the global `default.xml` layout handle for the theme. The `<script src="js/custom.js"/>` instruction belongs within the `<head>` section and resolves to the theme's `web/js/custom.js` file. Magento processes layout XML, publishes static view files as needed, and renders the resulting script reference in the page head for all pages that use the default layout handle. The file path is relative to the theme's `web` directory, so it should not include `web/` in the `src` value. Place this declaration in `app/design/frontend/<Vendor>/<theme>/Magento_Theme/layout/default.xml`; create the file when it does not already exist. This approach uses Magento's supported layout and static-asset resolution mechanisms rather than hard-coding a public URL in a template. Adobe's layout XML guidance documents adding JavaScript assets through the head configuration, and its theme structure documentation defines the theme `web` directory for static assets. See Adobe Commerce layout XML instructions and Adobe Commerce theme structure.

QUESTION NO: 13

You need to add a custom structural block template to a block that will render all children automatically. Which three attributes are required in the block's XML definition? (Choose THREE.)

A. after

B. as

C. before

D. name

E. template

F. type

ANSWER: B D F

Explanation:

For a Magento 1 structural block that is intended to act as a container and render its child blocks, the layout declaration needs **type**, **name**, and **as**. The **type** attribute identifies the Magento block model to instantiate. For a container that outputs its children, this is commonly a text-list block type or a custom block type that implements equivalent child rendering behavior. The **name** attribute gives the block its unique layout identifier, allowing later layout updates to reference, modify, or append content to that exact block. The **as** attribute assigns the child alias used by its parent block; this makes the structural block available through the parent's child-block relationship and is the layout handle by which parent templates

conventionally retrieve it. Together, these attributes establish the block's implementation, identity, and placement alias in the layout tree. Magento's layout system builds this tree from XML and uses child aliases when blocks are retrieved and rendered. See the Magento 1 layout XML overview in the Magento Developer Documentation and the layout XML guidance in the Adobe Commerce Developer Documentation.

QUESTION NO: 14

In Magento layouts, which attribute for defines the functionality of the block?

A. As

B. Action

C. Type

D. Name

ANSWER: C

Explanation:

The `type` attribute defines a Magento block's functionality because it identifies the block class, usually through a configured class alias. When Magento processes a layout XML instruction such as `<block type="catalog/product_view" ... />`, it resolves that type alias through configuration and instantiates the associated PHP block class. That class contains the behavior used to prepare data for the template, expose methods to the view layer, and participate in Magento's layout rendering process. In Magento's layout implementation, the block-generation flow reads the `type` value and passes it to the layout's block-creation mechanism, which is why this attribute determines what the block does rather than merely identifying where it appears. The Magento core layout code documents this creation path in Mage_Core_Model_Layout.php. Magento module configuration also maps block aliases to PHP class prefixes under the global blocks configuration, as illustrated in the core configuration structure at Mage_Catalog configuration.