NVIDIA NCP-OUSD Practice Questions

Shared by Hanson on 21-08-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

If you have a Usd.Prim object named my_prim and you want to specifically retrieve an attribute named "size", which method would you typically use?

Options:

A- my_prim.GetAttribute('size')
B- my_prim.GetRelationship('size')
C- my_prim.GetProperty('size')
D- my_prim.GetPrimvar('size')

Answer:

A

Explanation:

The correct method is my_prim.GetAttribute('size') because the question asks for a specific attribute, not a relationship or generic property. NVIDIA's Learn OpenUSD material defines properties as the data-bearing namespace objects on prims and distinguishes two property categories: attributes and relationships. Attributes hold typed values, while relationships point to other objects in the scene description. NVIDIA's Omniverse developer reference also states that Usd.Prim.GetAttribute() returns a Usd.Attribute, after which Usd.Attribute.Get() is used to resolve the actual authored or composed value.

Option A is therefore the precise API call. Option B is incorrect because GetRelationship() retrieves relationship properties, not value-bearing attributes. Option C is less specific: GetProperty('size') can retrieve either an attribute or relationship as a generic UsdProperty, requiring additional type handling. Option D is incorrect because primvars are accessed through schema-specific APIs such as UsdGeom.PrimvarsAPI, not directly as a general Usd.Prim method. This aligns with Pipeline Development USD Python API, Property Access, Attributes, Relationships, and Scenegraph Authoring.

Question 2

Question Type: MultipleChoice

When designing a scalable asset structure, which two aspects should be primarily considered?

Choose two.

Options:

A- Innovation and future-proofing
B- File formats and compression methods
C- Clients and collaborators
D- Modularity and performance

Answer:

C, D

Explanation:

A scalable asset structure should be designed around the needs of clients and collaborators and around structural principles such as modularity and performance. NVIDIA's Learn OpenUSD asset-structure guidance states that a well-designed scalable asset structure should always be tailored to the needs of clients and collaborators. It also emphasizes that collaboration is core to OpenUSD and that scalable structures should support parallel workstreams across multiple dimensions.

Option C is correct because asset structure is not created in isolation; it must serve the downstream consumers, artists, tools, departments, and delivery requirements that will interact with the asset. Option D is correct because NVIDIA identifies the four principles of scalable asset structure as Legibility, Modularity, Performance, and Navigability. Modularity enables reuse and flexible composition, while performance ensures efficient reads, writes, loading, and interaction at production scale.

Option A is too vague and not one of the named primary design aspects. Option B can matter tactically, but file formats and compression are implementation choices, not the primary asset-structure design drivers. This aligns with Content Aggregation Asset Structure Principles Clients, Collaborators, Modularity, Performance, and Scalability.

Question 3

Question Type: MultipleChoice

In OpenUSD, which USDA snippet correctly uses a payload to reference an external asset while allowing deferred loading?

Options:

A- def 'Character' (prepend payload = @character.usda@) { }
B- def 'Character' { prepend payloads = @character.usd@ }
C- def Xform 'Character' (reference = @character.usda@) { }
D- def Xform 'Character' (payload = @character.usd@) { }

Answer:

A

Explanation:

Option A is the correct USDA pattern because a payload is authored as prim metadata using the singular payload keyword with a list-editing operation such as prepend. NVIDIA's Learn OpenUSD payload exercise shows the canonical USDA form: prepend payload = @./red_cube.usd@, authored on the prim declaration. It further explains that payloads are similar to references in USDA, with the important distinction that the keyword is payload rather than references.

The key technical purpose of a payload is deferred loading. NVIDIA explains that payloads can compose scene description when loaded, or unload the targeted scene description beneath the payloaded prim. It also contrasts this with references, which are always composed and present on the stage, while payloads can be opened unloaded and selectively loaded later.

Option B is incorrect because payloads is not the USDA keyword and it is not authored as a property inside the prim body. Option C uses a reference, not a payload, so it does not provide payload load/unload behavior. Option D is not the canonical list-op form expected here. This aligns with Composition Reference and Payloads Working With Payloads.

Question 4

Question Type: MultipleChoice

Which option best methods allows you to edit the color of an instanceProxy mesh in OpenUSD while keeping the prim instanced?

Options:

A- Directly modifying the instance's geometry properties.
B- Using primvars to change the color of the mesh or assigned material.
C- Creating a relationship targeting the mesh's color attribute.
D- Replacing the entire instance with a new mesh that has the desired color.

Answer:

B

Explanation:

The correct method is to use primvars to drive shading variation while preserving instancing. NVIDIA's Learn OpenUSD scenegraph instance refinement guidance states that prototypes are runtime data and are not editable, instance proxies are not editable, and local opinions on instance proxies are discarded. It then identifies hierarchical refinement as the appropriate strategy for non-destructive instance variation, including inherited primvars. Primvars can drive material properties such as color, inherit down the prim hierarchy, and be authored on the instanceable prim or an ancestor so materials inside the instance subgraph can read the inherited value.

Option B is correct because it changes the effective appearance without directly editing the instance proxy mesh and without disabling instancing. Option A is incorrect because directly modifying an instance proxy's geometry properties is not allowed. Option C is incorrect because a relationship targeting a color attribute is not the standard mechanism for material color variation. Option D is incorrect because replacing the mesh defeats the purpose of preserving the instanced asset structure. This aligns with Content Aggregation Asset Modularity and Instancing Refining Scenegraph Instances, Hierarchical Refinement, Primvars, and Instance Editability.

Question 5

Question Type: MultipleChoice

In OpenUSD, when composing a stage, why might opinions from one layer not take effect in the final composed stage?

Options:

A- The opinions are in a non-editable layer that does not allow modifications during composition.
B- The layer containing the opinion does not use the correct schema for the data.
C- The layer is not referenced properly, so its opinions are ignored.
D- The layer containing the opinion has a lower strength than other layers, and its changes are overridden.

Answer:

D

Explanation:

An authored opinion may be valid and still not appear in the final composed stage because USD resolves competing opinions by strength ordering. NVIDIA's Learn OpenUSD strength-ordering guide explains that, for each prim and property, the composition engine evaluates opinions from contributing layers and composition arcs using LIVRPS, giving precedence to stronger opinions when conflicts occur. It also states that stronger layers can override or add to data defined in weaker layers, enabling non-destructive edits and overrides.

Option D is correct because a weaker layer's opinion can be hidden by a stronger opinion authored elsewhere in the layer stack or through a stronger composition context. This is a normal result of USD composition, not an error. Option A is incorrect because editability controls whether a layer can be modified, not whether its authored opinions can contribute during composition. Option B may indicate invalid or poorly modeled data, but schema choice alone is not the general reason an opinion loses. Option C can be a troubleshooting case for missing referenced content, but the question asks why opinions from a composed layer may not take effect. This aligns with Composition Layer Stacks, Opinion Strength, LIVRPS, Value Resolution, and Non-Destructive Overrides.

Question 6

Question Type: MultipleChoice

If you have a Usd.Prim object named my_prim and you want to specifically retrieve an attribute named "size", which method would you typically use?

Options:

A- my_prim.GetAttribute('size')
B- my_prim.GetRelationship('size')
C- my_prim.GetProperty('size')
D- my_prim.GetPrimvar('size')

Answer:

A

Explanation:

The correct method is my_prim.GetAttribute('size') because the question asks for a specific attribute, not a relationship or generic property. NVIDIA's Learn OpenUSD material defines properties as the data-bearing namespace objects on prims and distinguishes two property

categories: attributes and relationships. Attributes hold typed values, while relationships point to other objects in the scene description. NVIDIA's Omniverse developer reference also states that Usd.Prim.GetAttribute() returns a Usd.Attribute, after which Usd.Attribute.Get() is used to resolve the actual authored or composed value.

Option A is therefore the precise API call. Option B is incorrect because GetRelationship() retrieves relationship properties, not value-bearing attributes. Option C is less specific: GetProperty('size') can retrieve either an attribute or relationship as a generic UsdProperty, requiring additional type handling. Option D is incorrect because primvars are accessed through schema-specific APIs such as UsdGeom.PrimvarsAPI, not directly as a general Usd.Prim method. This aligns with Pipeline Development USD Python API, Property Access, Attributes, Relationships, and Scenegraph Authoring.

Question 7

Question Type: MultipleChoice

Which option best are valid reasons for choosing to use or develop a standalone converter instead of an importer/exporter for a digital content creation (DCC) application? Choose two.

Options:

A- Standalone converters are the only way to convert proprietary formats.
B- A converter always produces higher-fidelity results compared to an importer/exporter.
C- You don't have access to an SDK to develop a native exporter for the DCC.
D- Your workflow requires batch conversion that is not suitable within the DCC.

Answer:

C, D

Explanation:

A standalone converter is appropriate when the conversion workflow should operate outside the DCC application. NVIDIA's Learn OpenUSD Data Exchange lesson defines a standalone converter as a script, executable, or microservice dedicated to translating to and from another file format. It contrasts this with importers and exporters, which are implemented as part of a DCC application's runtime or plugin system. (docs.nvidia.com)

Option C is correct because if a developer cannot access the DCC's SDK, plugin API, or runtime integration path, a native importer/exporter may not be feasible. A separate converter can still be built around accessible file-level data, command-line tools, services, or external APIs. Option D is also correct because standalone converters are well suited for automation, headless

processing, farm execution, and batch conversion pipelines that do not belong inside an artist-facing DCC session. NVIDIA's OpenUSD guidance also notes that data exchange implementations must account for different client needs, workflow performance, and content structures rather than assuming one implementation pattern fits all. (docs.nvidia.com)

Option A is false because proprietary formats can also be supported by native plugins or file format plugins when appropriate access exists. Option B is false because fidelity depends on data mapping and implementation quality, not on whether the tool is standalone. This aligns with Data Exchange Importers, Exporters, Standalone Converters, File Format Plugins, and Workflow Requirements.