NVIDIA
NCP-OUSD
NVIDIA-Certified Professional: OpenUSD Development (NCP-OUSD)

Product Version

✓ Up to Date products, reliable and verified.
✓ Questions and Answers in PDF Format.

Latest Version: 6.0

Question: 1

Which of the following statements best describes the purpose of OpenUSD file format plugins?

A. They extend OpenUSD's functionality by allowing it to read and write from various file formats.
B. They are only used for visualizing OpenUSD data and geometry in 3D applications.
C. They convert OpenUSD files to other formats without any loss of data or information.
D. They are designed to compress OpenUSD asset files for faster loading times.

Answer: A

Explanation:
OpenUSD file format plugins belong under the Data Exchange topic because they expand how USD participates in interchange workflows. Their purpose is to allow OpenUSD to read, interpret, and in some cases write data using formats beyond the native USD file types such as .usd, .usda, .usdc, and .usdz. Through these plugins, external file formats can be exposed to USD as layers and can participate in composition arcs such as references, payloads, and sublayers. This allows pipelines to integrate heterogeneous asset sources while still using USD's scene description model, composition engine, and layer-based workflows.
Option A is correct because it directly identifies the function of file format plugins: extending OpenUSD functionality for reading and writing various file formats. Option B is incorrect because visualization is only one possible downstream use of exchanged data, not the purpose of the plugin system. Option C is incorrect because translation between formats does not inherently guarantee lossless preservation of every schema, opinion, or semantic construct. Option D is incorrect because compression is not the primary role of file format plugins. This aligns with the NVIDIA OpenUSD Development Study Guide topic Data Exchange, especially file formats, connectors, and plugin-based interoperability.

Question: 2

You are a developer creating an OpenUSD exporter for an application that also supports import of USD assets. To enable collaborative workflows, you're adding an "export as overrides" option. Which approach correctly describes which structure your exporter should generate?

A. Overs of the prims and properties that have been modified or added, omitting unchanged data.

B. Export each prim separately into multiple layers, and reference them individually to maintain sparsity.
C. Include explicit definitions of all prims, properties, and relationships exactly matching the imported asset to ensure consistency.

Answer: A

Explanation:
The correct exporter behavior is to generate sparse overrides that represent only the authored contribution of the current workstream. In OpenUSD data exchange workflows, an exporter should not blindly rewrite the full imported asset when the intent is to preserve collaborative, non-destructive editing. Instead, the exporter should author only the changes required to express the current application's contribution: modified prims, newly added prims, changed attributes, relationships, metadata, or other authored opinions.
Option A is correct because an over is specifically used to contribute opinions to an existing prim without redefining the entire prim structure. This allows the exported layer to sit above the source asset in a layer stack and override only the relevant data. Option B incorrectly equates sparsity with splitting each prim into separate referenced layers; USD sparsity is achieved by authoring minimal opinions, not by unnecessary layer fragmentation. Option C is incorrect because exporting full definitions for every prim and property would duplicate unchanged data, reduce clarity, and undermine USD's composition-based collaboration model. This maps to the NVIDIA OpenUSD Development Study Guide topics Data Exchange, especially exporter design, data transformation, sparse authoring, and collaborative layer-based workflows.

Question: 3

In what way do variant sets in OpenUSD enhance flexibility in scene descriptions?

A. By allowing runtime selection among alternative representations of a prim.
B. By permanently embedding multiple scene configurations within a single prim.
C. By enabling automatic resolution of conflicting opinions across layers.
D. By statically merging all possible variants into one combined representation.

Answer: A

Explanation:
Variant sets enhance flexibility by allowing a prim to expose named alternatives, where one variant selection contributes its authored opinions to the composed result. NVIDIA's Learn OpenUSD material describes variant sets as a way to define "alternative representations for a prim and switch between them without duplicating data." It also explains that a prim may have one or more named variant sets, each containing variant choices, and that the selected variant composes the opinions authored for that choice.
Option A is correct because variant sets support selectable alternatives such as different model shapes, material looks, levels of detail, configurations, or composition arcs. This gives

downstream tools or stronger layers the ability to choose a representation non-destructively without rewriting the asset. Option B is inaccurate because variants are not permanently merged into the prim; only the selected variant participates in the composed scene. Option C is incorrect because conflict resolution is handled by USD's composition and value-resolution rules, not automatically by variant sets themselves. Option D is also incorrect because USD does not statically combine every variant into a single representation. This aligns with the NVIDIA OpenUSD Development Study Guide topics Composition → Variant Sets, Composition Arcs, and LIVERPS strength ordering.

Question: 4

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

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: 5

Which of these are valid types for custom attributes in OpenUSD? (Choose two.)

A. structs

B. assset paths
C. string arrays
D. dictionaries

Answer: B, C

Explanation:
Custom attributes in OpenUSD are user-defined properties authored on prims to store additional typed data beyond predefined schema attributes. NVIDIA's Learn OpenUSD material states that custom attributes can hold "numeric values, strings, or arrays," and that custom attributes are created with UsdPrim::CreateAttribute(), where Sdf.ValueTypeNames represents the attribute type. NVIDIA's OpenUSD data type reference lists Asset with the USDA value type token asset, and StringArray with the USDA token string[], making asset paths and string arrays valid attribute value types.
Option B is correct because asset path values are represented through the asset value type and are commonly used for resource references such as textures or external asset identifiers. Option C is correct because string[] is a valid array value type. Option A is incorrect because USD does not have a native struct attribute type; NVIDIA recommends namespace-prefixed attributes for mapping grouped or compound fields. Option D is incorrect in this context because dictionaries are associated with metadata/customData patterns, not ordinary typed custom attributes. This aligns with Customizing USD → Custom Properties, Sdf.ValueTypeNames, Namespaced Attributes.

Thank You for Trying Our Product

For More Information – Visit link below:

https://www.examsboost.com/

15 USD Discount Coupon Code:

G74JA8UF

FEATURES

- ✓ 90 Days Free Updates
- ✓ Money Back Pass Guarantee
- ✓ Instant Download or Email Attachment
- ✓ 24/7 Live Chat Support
- ✓ PDF file could be used at any Platform
- ✓ 50,000 Happy Customer