***Q1.***   
1. Blender provides an API that can be interacted with using Python. How can you use Python scripting to automate the creation of a 3D model in Blender? Please provide a basic code example.  
 ***Answer to Question 1  
1.*** To use Python scripting to automate the creation of a 3D model in Blender, you can utilize the Blender Python API (bpy). Here's a basic code example that demonstrates how to create a triangle.  
import bpy

# Clear existing objects

bpy.ops.object.select\_all(action='DESELECT')

bpy.ops.object.select\_by\_type(type='MESH')

bpy.ops.object.delete()

# Create a new mesh object

mesh = bpy.data.meshes.new(name="TriangleMesh")

obj = bpy.data.objects.new("Triangle", mesh)

# Link the object to the scene

scene = bpy.context.scene

scene.collection.objects.link(obj)

# Create vertices and faces for the triangle

verts = [

(0, 1, 0), # Vertex 0

(-1, -1, 0), # Vertex 1

(1, -1, 0) # Vertex 2

]

faces = [

(0, 1, 2) # Face with vertices 0, 1, 2

]

# Update the mesh with the new data

mesh.from\_pydata(verts, [], faces)

mesh.update()

# Center the triangle in the scene

obj.location = (0, 0, 0)

# Render the scene

bpy.ops.render.render(write\_still=True)

**2**. In Blender's Python API, what is the purpose of the bpy module? How can you use it to manipulate object transformations in a 3D scene?  
  
**Ans:**  
In Blender's Python API, the **bpy** module is the main module that provides access to Blender's functionality and allows you to interact with the Blender application using Python scripts. It serves as a gateway to various Blender data and operations.  
To manipulate object transformations in a 3D scene using the **bpy** module, you can follow these steps:  
Import the bpy module:  
 import bpy

Access the active scene:  
 scene = bpy.context.scene

Access the objects in the scene:  
 objects = bpy.data.objects

Select the object(s) you want to manipulate:

# Select all objects

for obj in objects:

obj.select\_set(True)

# Select specific object by name

object\_name = "Triangle"

objects[object\_name].select\_set(True)

Modify the object's transformations:

# Access the active object  
active\_object = bpy.context.active\_object

# Translate (move) the object  
active\_object.location = (x, y, z)

# Rotate the object   
active\_object.rotation\_euler = (rx, ry, rz) # in radians

# Scale the object  
active\_object.scale = (sx, sy, sz)

Q2.1: Describe the steps to create a Docker container for a Python-based application. What information would you need to include in the Dockerfile?  
  
Ans:  
-Create a file named "Dockerfile" (without any file extension) in your project directory.  
-Open the Dockerfile in a text editor and specify the following information:

1. Use the chosen Python base image as the starting point  
   FROM python:<python\_version>
2. Set the working directory inside the container:  
   WORKDIR /app
3. Copy the application files into the container:  
   COPY . /app
4. Install any required dependencies using pip:  
   RUN pip install -r requirements.txt
5. Expose any necessary ports if your application listens on a specific port:  
   EXPOSE <port\_number>
6. Specify the command to run when the container starts:  
   CMD ["python3", "app.py"]

-Save and close the Dockerfile.  
-Open a command-line interface, navigate to the project directory, and build the Docker image using the following command:

docker build -t <image\_name> .

-Once the image is built, you can run a container using the created image:

docker run -d -p <host\_port>:<container\_port> <image\_name>

**Q2.2: Explain how you can use Docker Compose to manage multi-container Python applications.  
Ans:** I don't have knowledge to provide an accurate answer to this question.