 # 01 Install OpenCV and PyTorch (verify imports)

Description : On your Linux system use `pip` to install `opencv-python` and `torch`, then open Python and run `import cv2, torch` to confirm both import without errors.
Run a short snippet that loads an image with `cv2.imread`, converts it to a `torch` tensor, and prints the tensor shape.

 # 02 Basic image processing with OpenCV

Description : Write a short Python script that loads an image with `cv2.imread`, converts it to grayscale using `cv2.cvtColor`, and displays both images with `cv2.imshow` (or saves them if display is unavailable).
Save the grayscale image to disk and print the original and grayscale image shapes and dtypes to the console.

 # 03 Run a pretrained ResNet on a single image

Description : Using `torchvision` or `torch.hub`, load a pretrained ResNet (e.g., `resnet18`), apply the standard resize/center-crop/ToTensor/normalize preprocessing, and run a forward pass on one image.
Compute softmax probabilities and print the top-3 predicted class indices and probabilities (run on CPU if a GPU isn't available).

 # 04 Install PCL & Eigen and compile a minimal C++ program

Description : On Linux install `libpcl-dev` and `libeigen3-dev` with `sudo apt`, create a tiny C++ file that includes `<pcl/point_types.h>` and `<Eigen/Dense>`, then construct a `pcl::PointXYZ` and an `Eigen::Vector3d`.
Compile with `g++` specifying include paths (e.g., `-I/usr/include/pcl-1.12 -I/usr/include/eigen3`), run the binary, and print the point's coordinates to stdout.

 # 05 Create a ROS 2 Python node: camera → grayscale → model label

Description : Create a minimal ROS 2 Python package (with `rclpy`, `sensor_msgs`, `cv_bridge`) that subscribes to `sensor_msgs/Image`, converts to OpenCV via CvBridge, converts to grayscale, runs a pretrained torch model to infer a label, and publishes the label as `std_msgs/String`.
Build the workspace with `colcon build`, run the node, and verify outputs using `ros2 topic echo` on the label topic.