```
/**
@defgroup python_skeletontracking_console_img Skeleton estimation on a local
image
@ingroup python_samples
\section Introduction
The Skeleton Tracking Python binding provides a convenient interface to perform
the following tasks:
- Load the DNN model onto the target compute device (currently Intel CPU and
Intel GPU are supported).
- Perform inference on the input image (currently only 3 channel 8-bit images of
BGR format are supported)
- Draw the estimated keypoints of the skeleton over the input image.

\section Pre-requisites
- Currently this application supports an Intel CPU as well as GPU options with
an 18-point Skeleton Tracking model. The model should match the
TargetComputeDevice: fp32 for CPU and fp16 or fp32 for GPU
- The models need to be in the default models directory. The default models
directory on Windows is %%LOCALAPPDATA%\\Cubemos\\SkeletonTracking\\models

\section Explanation
The python sample uses argparse to generate a command line interface. You can
call "python estimate-keypoints --help" to see all available options.
The main method looks as follows:
\code{.py}
if __name__ == "__main__":
    #Parse command line arguments
    args = parser.parse_args()
    #Get the path of the native libraries and ressource files
    sdk_path = os.environ["CUBEMOS_SKEL_SDK"]
    if args.verbose:
        initialise_logging(sdk_path, CM_LogLevel.CM_LL_DEBUG, True)
    img = cv2.imread(args.image)
    #initialize the api with a valid license key in default_license_dir()
    api = Api(default_license_dir())
    model_path = os.path.join(
        sdk_path, "models", "skeleton-tracking", "fp32", "skeleton-
tracking.cubemos"
    )
    api.load_model(CM_TargetComputeDevice.CM_CPU, model_path)
    #perform inference
    skeletons = api.estimate_keypoints(img, 192)
    render_result(skeletons, img, args.confidence_threshold)

    cv2.imwrite(args.output_image, img)
\endcode

In order to get started with using Skeleton Tracking on saved images and cubemos
python wrapper, check the full source code of this sample provided in
$CUBEMOS_SKEL_SDK/samples/python/estimate-keypoints.py on linux and \
%CUBEMOS_SKEL_SDK%\samples\python\estimate-keypoints.py on windows. \n


@defgroup python_skeletontracking_realsense Skeleton tracking using Intel&reg;
RealSense&trade;
@ingroup python_samples
\section Use-cases
The purpose of this sample is to show three use-cases
- How to acquire rgb and depth streams of an Intel&reg; RealSense&trade; camera.
- How to do Skeleton Tracking on the rgb data.
- How to get the corresponding 3D locations of the skeleton joints using
realsense depth data.
```

\section Pre-requisites
- Currently this application supports an Intel CPU as well as GPU options with an 18-point Skeleton Tracking model. The model should match the TargetComputeDevice: fp32 for CPU and fp16 or fp32 for GPU
- The models need to be in the default models directory. The default models directory on Windows is %%LOCALAPPDATA%\\Cubemos\\SkeletonTracking\\models


\section Explanation
The python wrapper of the <a href=https://github.com/IntelRealSense/librealsense>librealsense</a> library is used
to set up the RGB and Depth streams. For more information on how to use the python wrapper
please see the <a href=https://github.com/IntelRealSense/librealsense/tree/master/wrappers/python>respective documentation</a>.

\code{.py}
```
if __name__ == "__main__":
    try:
        # Configure depth and color streams of the intel realsense
        config = rs.config()
        config.enable_stream(rs.stream.depth, 1280, 720, rs.format.z16, 30)
        config.enable_stream(rs.stream.color, 1280, 720, rs.format.bgr8, 30)

        # Start the realsense pipeline
        pipeline = rs.pipeline()
        pipeline.start()

        # Get the intrinsics information for calculation of 3D point
        frames = pipeline.wait_for_frames()
        depth = frames.get_depth_frame()
        depth_intrinsic = depth.profile.as_video_stream_profile().intrinsics

        # Initialize the cubemos api with a valid license key in default_license_dir()
        skeletrack = skeletontracker(cloud_tracking_api_key = "")
        joint_confidence = 0.2

        # Create window for initialisation
        window_name = "cubemos skeleton tracking with realsense D400 series"
        cv2.namedWindow(window_name, cv2.WND_PROP_FULLSCREEN)
        cv2.setWindowProperty(window_name, cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)

        while True:
            # Create a pipeline object. This object configures the streaming camera and owns it's handle
            frames = pipeline.wait_for_frames()
            depth = frames.get_depth_frame()
            color = frames.get_color_frame()
            if not depth or not color: continue

            # Convert images to numpy arrays
            depth_image = np.asanyarray(depth.get_data())
            color_image = np.asanyarray(color.get_data())
```

\endcode

The actual skeleton tracking happens in the few lines of code below.
At first the skeleton estimation is done on the color image and shown as an overlay.
Afterwards, in `render_ids_3d` the corresponding 3d coordinates to to 2d joints

```
are
calculated and also shown as an overlay in the color image.
\code{.py}
# perform inference and update the tracking id
skeletons = skeletrack.track_skeletons(color_image)

# render the skeletons on top of the acquired image and display it
color_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2RGB)
cm.render_result(skeletons, color_image, joint_confidence)
render_ids_3d(color_image, skeletons, depth, depth_intrinsic, joint_confidence)
cv2.imshow(window_name, color_image)
\endcode

@defgroup python_skeletontracking_webcam Skeleton tracking using the webcam
@ingroup python_samples
\section Use-cases
The purpose of this sample is to show three use-cases
- How to acquire an rgb data stream from the webcam using python and opencv.
- How to do Skeleton Tracking on the rgb data.
- How to use cloud-tracking to increase the tracking accuracy.


\section Pre-requisites
- Currently this application supports an Intel CPU as well as GPU options with
an 18-point Skeleton Tracking model. The model should match the
TargetComputeDevice: fp32 for CPU and fp16 or fp32 for GPU
- The models need to be in the default models directory. The default models
directory on Windows is %%LOCALAPPDATA%\\Cubemos\\SkeletonTracking\\models
- To use cloud-tracking you need a valid api-key. If you do not currently pocess
one, please visit https://console.cubemos.com to generate one.

\section Code
\code{.py}
#!/usr/bin/env python3
from skeletontracker import skeletontracker
import util as cm
import cv2

# entry point
if __name__ == "__main__":
    try:
        cloud_tracking_api_key = cm.get_cloud_tracking_api_key()

        # initialise the webcam
        capture_device = cv2.VideoCapture(0)
        hasFrame, frame = capture_device.read()

        # initialize the cubemos skeleton tracking
        skeletrack = skeletontracker(cloud_tracking_api_key)
        joint_confidence = 0.2

        # create window for initialisation
        window_name = "cubemos skeleton tracking with webcam as input source"
        cv2.namedWindow(window_name, cv2.WND_PROP_FULLSCREEN)
        cv2.setWindowProperty(window_name, cv2.WND_PROP_FULLSCREEN,
cv2.WINDOW_FULLSCREEN)

        while hasFrame:
            # perform inference and update the tracking id
            skeletons = skeletrack.track_skeletons(frame)

            # render the skeletons on top of the acquired image and display it
            cm.render_result(skeletons, frame, joint_confidence)
            cm.render_ids(skeletons, frame)
```

```
            # show the result on on opencv window
            cv2.imshow(window_name, frame)
            if cv2.waitKey(1) == 27:
                break;

            # Capture a new frame
            hasFrame, frame = capture_device.read()

    except Exception as ex:
        print("Exception occured: \"{}\"".format(ex))
\endcode

*/
```