```
!pip install mediapipe
```

```
Collecting mediapipe
  Downloading mediapipe-0.10.18-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_6
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: attrs>=19.1.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: jax in /usr/local/lib/python3.10/dist-packages (from m
Requirement already satisfied: jaxlib in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy<2 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: opencv-contrib-python in /usr/local/lib/python3.10/dis
Requirement already satisfied: protobuf<5,>=4.25.3 in /usr/local/lib/python3.10/dist-
Collecting sounddevice>=0.4.4 (from mediapipe)
  Downloading sounddevice-0.5.1-py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: sentencepiece in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: CFFI>=1.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: ml-dtypes>=0.2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scipy>=1.10 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (f
Downloading mediapipe-0.10.18-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 36.1/36.1 MB 44.1 MB/s eta 0:00:00
Downloading sounddevice-0.5.1-py3-none-any.whl (32 kB)
Installing collected packages: sounddevice, mediapipe
Successfully installed mediapipe-0.10.18 sounddevice-0.5.1
```

FACEMESH

```
import cv2
import mediapipe as mp
import time
from IPython.display import Video

class FaceMeshDetector:
    def __init__(self, staticMode=False, maxFaces=2, minDetectionCon=0.5, minTrackCon=0.5
        self.staticMode = staticMode
        self.maxFaces = maxFaces
        self.minDetectionCon = minDetectionCon
        self.minTrackCon = minTrackCon
        self.mpDraw = mp.solutions.drawing_utils
        self.mpFaceMesh = mp.solutions.face_mesh
        self.faceMesh = self.mpFaceMesh.FaceMesh(
```

```
                static_image_mode=self.staticMode,
                max_num_faces=self.maxFaces,
                min_detection_confidence=self.minDetectionCon,
                min_tracking_confidence=self.minTrackCon
            )
            self.drawSpec = self.mpDraw.DrawingSpec(thickness=1, circle_radius=2)

        def findFaceMesh(self, img, draw=True):
            imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            results = self.faceMesh.process(imgRGB)
            faces = []
            if results.multi_face_landmarks:
                for faceLms in results.multi_face_landmarks:
                    if draw:
                        self.mpDraw.draw_landmarks(
                            img, faceLms, mp.solutions.face_mesh.FACEMESH_CONTOURS,
                            self.drawSpec, self.drawSpec)
                    face = []
                    for lm in faceLms.landmark:
                        ih, iw, ic = img.shape
                        x, y = int(lm.x * iw), int(lm.y * ih)
                        face.append([x, y])
                    faces.append(face)
            return img, faces

    def main():
        # Load video file
        cap = cv2.VideoCapture("/content/drive/MyDrive/WIN_20241025_12_47_03_Pro.mp4")
        pTime = 0
        detector = FaceMeshDetector(maxFaces=2)

        # Define video writer for output
        frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = cap.get(cv2.CAP_PROP_FPS)
        out = cv2.VideoWriter('output_with_mesh.mp4', cv2.VideoWriter_fourcc(*'mp4v'), fps, (

        while cap.isOpened():
            success, img = cap.read()
            if not success:
                break

            img, faces = detector.findFaceMesh(img)
            if faces:
                print(faces[0])  # Print the first detected face landmarks for reference

            # Calculate FPS
            cTime = time.time()
            fps = 1 / (cTime - pTime) if (cTime - pTime) != 0 else 0
            pTime = cTime
            cv2.putText(img, f'FPS: {int(fps)}', (20, 70), cv2.FONT_HERSHEY_PLAIN,
                        3, (0, 255, 0), 3)

            # Write each frame to the output video
            out.write(img)
```

```
    cap.release()
    out.release()
    print("Video processing complete. Saved as output_with_mesh.mp4")

# Run main to process the video
main()

# Display the output video
Video("output_with_mediapipe_mesh.mp4", embed=True)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-3-7737fb9a3e36> in <cell line: 2>()
      1 import cv2
----> 2 import mediapipe as mp
      3 import time
      4 from IPython.display import Video
      5

ModuleNotFoundError: No module named 'mediapipe'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

FACELANDMARK

```
from google.colab import drive
import cv2
import mediapipe as mp
import time
from IPython.display import Video, display

# Mount Google Drive
drive.mount('/content/drive')

class FaceMeshDetector:
    def __init__(self, staticMode=False, maxFaces=2, minDetectionCon=0.5, minTrackCon=0.5
        self.staticMode = staticMode
        self.maxFaces = maxFaces
        self.minDetectionCon = minDetectionCon
        self.minTrackCon = minTrackCon
        self.mpDraw = mp.solutions.drawing_utils
```

```
            self.mpFaceMesh = mp.solutions.face_mesh
            self.faceMesh = self.mpFaceMesh.FaceMesh(
                static_image_mode=self.staticMode,
                max_num_faces=self.maxFaces,
                min_detection_confidence=self.minDetectionCon,
                min_tracking_confidence=self.minTrackCon
            )
            self.drawSpec = self.mpDraw.DrawingSpec(thickness=1, circle_radius=2)

        def findFaceMesh(self, img, draw=True, highlight_count=0):
            imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            results = self.faceMesh.process(imgRGB)
            faces = []
            if results.multi_face_landmarks:
                for faceLms in results.multi_face_landmarks:
                    face = []
                    for id, lm in enumerate(faceLms.landmark):  # Use all landmarks
                        ih, iw, ic = img.shape
                        x, y = int(lm.x * iw), int(lm.y * ih)
                        face.append([x, y])

                        if draw:

                            if id < highlight_count:
                                cv2.circle(img, (x, y), 4, (0, 0, 255), -1)  # Highlight colo
                            else:
                                cv2.circle(img, (x, y), 2, (0, 255, 0), -1)  # Regular color
                    faces.append(face)
            return img, faces

    def main():
        # Ask the user for the number of landmarks to highlight
        try:
            highlight_count = int(input("Enter the number of landmarks to highlight : "))
            if highlight_count > 468:  # Update according to MediaPipe Face Mesh total landma
                raise ValueError("Number of landmarks cannot exceed 468.")
        except ValueError:
            print("Invalid input. Using default value of 68 landmarks.")
            highlight_count = 68  # Default value if input is invalid

        # Load your video file
        cap = cv2.VideoCapture("/content/drive/MyDrive/WIN_20241029_12_25_48_Pro (1).mp4")  #
        pTime = 0
        detector = FaceMeshDetector(maxFaces=2)

        # Define video writer for output - specify Google Drive path
        output_path = '/content/drive/MyDrive/output_with_Highlighted_landmark.mp4'  # Change
        frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = cap.get(cv2.CAP_PROP_FPS)
        out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (frame_width

        while cap.isOpened():
            success, img = cap.read()
            if not success:
```

```
            break

        # Process each frame with the specified number of landmarks
        img, faces = detector.findFaceMesh(img, highlight_count=highlight_count)
        if faces:
            print(faces[0])  # Print the first detected face landmarks for reference

        # Calculate FPS
        cTime = time.time()
        fps = 1 / (cTime - pTime) if (cTime - pTime) != 0 else 0
        pTime = cTime
        cv2.putText(img, f'FPS: {int(fps)}', (20, 70), cv2.FONT_HERSHEY_PLAIN,
                    3, (0, 255, 0), 3)

        # Write each frame to the output video
        out.write(img)

    cap.release()
    out.release()
    print(f"Video processing complete. Saved to Google Drive as {output_path}")

# Run main to process the video
main()

# Display the output video from Google Drive
display(Video('/content/drive/MyDrive/output_with_Highlighted_landmark.mp4', embed=True))
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
Enter the number of landmarks to highlight : 50
[[1698, 133], [1685, 102], [1695, 109], [1678, 65], [1684, 92], [1685, 77], [1689, 36
[[1668, 121], [1651, 93], [1664, 100], [1644, 59], [1648, 83], [1649, 69], [1654, 34]
[[1643, 116], [1623, 90], [1639, 95], [1619, 52], [1620, 80], [1621, 64], [1630, 24],
Video processing complete. Saved to Google Drive as /content/drive/MyDrive/output_wit
```

FACELANDMESH_WITH_GIVEN_LANDMARK_POINTS

```
import cv2
import mediapipe as mp
import time
from IPython.display import Video
from google.colab import drive
```

```
    # Mount Google Drive
    drive.mount('/content/drive')

    class FaceMeshDetector:
        def __init__(self, staticMode=False, maxFaces=2, minDetectionCon=0.5, minTrackCon=0.5
            self.staticMode = staticMode
            self.maxFaces = maxFaces
            self.minDetectionCon = minDetectionCon
            self.minTrackCon = minTrackCon
            self.mpDraw = mp.solutions.drawing_utils
            self.mpFaceMesh = mp.solutions.face_mesh
            self.faceMesh = self.mpFaceMesh.FaceMesh(
                static_image_mode=self.staticMode,
                max_num_faces=self.maxFaces,
                min_detection_confidence=self.minDetectionCon,
                min_tracking_confidence=self.minTrackCon
            )
            self.drawSpec = self.mpDraw.DrawingSpec(thickness=1, circle_radius=2)

        def findFaceMesh(self, img, highlight_count=0):
            imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            results = self.faceMesh.process(imgRGB)
            faces = []
            if results.multi_face_landmarks:
                for faceLms in results.multi_face_landmarks:
                    # Draw the complete mesh (optional)
                    self.mpDraw.draw_landmarks(
                        img, faceLms, mp.solutions.face_mesh.FACEMESH_CONTOURS,
                        self.drawSpec, self.drawSpec)

                    face = []
                    for id, lm in enumerate(faceLms.landmark):
                        ih, iw, ic = img.shape
                        x, y = int(lm.x * iw), int(lm.y * ih)
                        face.append([x, y])

                        # Highlight only specified landmarks
                        if id < highlight_count:
                            cv2.circle(img, (x, y), 4, (0, 0, 255), -1)  # Highlight color (r
                    faces.append(face)
            return img, faces

    def main():
        # Get user input for number of landmarks to highlight
        try:
            highlight_count = int(input("Enter the number of landmarks to highlight : "))
            highlight_count = min(highlight_count, 468)  # Ensure it does not exceed 468
        except ValueError:
            print("Invalid input. Using default value of 68 landmarks.")
            highlight_count = 68  # Default value if input is invalid

        # Load your video file
        cap = cv2.VideoCapture("/content/drive/MyDrive/WIN_20241029_12_46_11_Pro.mp4")
        pTime = 0
        detector = FaceMeshDetector(maxFaces=2)
```

```
    # Define video writer for output
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)
    output_path = '/content/drive/MyDrive/output_with_given_landmarks.mp4'  # Specify the
    out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (frame_width

    while cap.isOpened():
        success, img = cap.read()
        if not success:
            break

        img, faces = detector.findFaceMesh(img, highlight_count)
        if faces:
            print(faces[0])  # Print the first detected face landmarks for reference

        # Calculate FPS
        cTime = time.time()
        fps = 1 / (cTime - pTime) if (cTime - pTime) != 0 else 0
        pTime = cTime
        cv2.putText(img, f'FPS: {int(fps)}', (20, 70), cv2.FONT_HERSHEY_PLAIN,
                    3, (0, 255, 0), 3)

        # Write each frame to the output video
        out.write(img)

    cap.release()
    out.release()
    print(f"Video processing complete. Saved as {output_path}")

# Run main to process the video
main()

# Display the output video
Video("/content/drive/MyDrive/output_with_given_landmarks.mp4", embed=True)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
Enter the number of landmarks to highlight : 68
[[814, 723], [805, 702], [814, 709], [805, 680], [804, 695], [806, 686], [813, 663
[[808, 718], [798, 699], [807, 705], [799, 678], [798, 693], [800, 685], [808, 664
[[800, 705], [791, 686], [799, 692], [791, 665], [791, 680], [793, 671], [800, 651
[[785, 702], [775, 682], [784, 688], [777, 662], [774, 676], [777, 668], [786, 648
[[770, 705], [758, 685], [769, 691], [760, 665], [757, 679], [760, 671], [770, 651
[[755, 716], [741, 696], [754, 702], [743, 675], [739, 690], [742, 682], [753, 661
[[742, 732], [729, 712], [742, 718], [732, 691], [727, 706], [730, 698], [741, 675
[[725, 739], [715, 721], [726, 726], [718, 702], [714, 715], [717, 708], [728, 688
[[714, 748], [703, 729], [715, 735], [706, 712], [702, 724], [705, 717], [715, 699
[[698, 749], [684, 732], [697, 737], [688, 713], [683, 727], [686, 719], [697, 699
[[684, 743], [668, 725], [683, 731], [673, 705], [667, 719], [670, 711], [683, 690
[[678, 735], [663, 715], [677, 722], [668, 695], [662, 709], [665, 701], [678, 682
[[675, 720], [658, 700], [673, 707], [663, 679], [657, 694], [661, 686], [674, 665
[[672, 709], [656, 688], [671, 694], [661, 666], [655, 681], [658, 672], [672, 650
[[675, 700], [660, 678], [674, 685], [664, 656], [659, 671], [662, 663], [674, 641
[[677, 695], [662, 671], [676, 678], [666, 650], [661, 665], [664, 656], [676, 635
```

```
[[679, 696], [666, 673], [678, 680], [669, 652], [665, 667], [668, 658], [679, 638
[[681, 703], [668, 679], [681, 687], [671, 658], [667, 673], [670, 664], [681, 644
[[687, 715], [677, 689], [687, 697], [679, 668], [676, 682], [679, 674], [689, 654
[[696, 727], [687, 701], [696, 709], [688, 680], [686, 694], [689, 686], [698, 660
[[704, 737], [696, 712], [704, 720], [697, 690], [695, 705], [698, 696], [707, 676
[[712, 745], [703, 718], [712, 726], [703, 695], [703, 711], [705, 702], [714, 680
[[717, 741], [707, 715], [716, 723], [708, 692], [707, 708], [709, 699], [717, 677
[[720, 734], [709, 708], [718, 716], [709, 685], [708, 701], [710, 692], [718, 669
[[713, 720], [704, 694], [712, 702], [704, 671], [703, 687], [705, 678], [714, 655
[[698, 700], [688, 676], [698, 683], [689, 652], [687, 669], [690, 659], [701, 637
[[684, 684], [672, 660], [683, 668], [674, 637], [671, 653], [674, 644], [685, 622
[[669, 681], [660, 651], [671, 661], [662, 622], [659, 642], [662, 631], [675, 602
[[657, 673], [645, 648], [656, 656], [647, 624], [644, 641], [647, 631], [658, 609
[[650, 678], [642, 651], [652, 660], [641, 622], [641, 642], [643, 631], [652, 602
[[641, 685], [629, 660], [640, 667], [630, 636], [628, 652], [631, 643], [640, 620
[[638, 699], [628, 671], [637, 679], [626, 647], [627, 663], [629, 653], [636, 630
[[629, 712], [620, 686], [628, 694], [618, 662], [619, 678], [621, 668], [628, 645
[[618, 729], [613, 700], [619, 709], [611, 672], [612, 691], [614, 680], [621, 653
[[598, 728], [588, 702], [596, 710], [586, 677], [588, 694], [589, 684], [596, 660
[[580, 726], [571, 700], [579, 708], [568, 675], [570, 692], [571, 682], [577, 658
[[564, 719], [554, 693], [562, 701], [551, 667], [553, 685], [555, 675], [560, 649
[[553, 707], [543, 681], [551, 688], [539, 655], [541, 673], [543, 662], [548, 637
[[542, 695], [532, 667], [540, 675], [529, 641], [531, 659], [532, 648], [538, 622
[[535, 682], [527, 655], [534, 663], [522, 628], [526, 646], [526, 636], [531, 609
[[527, 677], [517, 649], [525, 658], [513, 623], [516, 641], [517, 631], [522, 604
[[518, 680], [509, 651], [516, 660], [505, 625], [508, 643], [509, 632], [513, 606
[[509, 690], [500, 660], [507, 669], [496, 633], [499, 651], [500, 641], [505, 615
[[501, 707], [493, 676], [499, 685], [488, 649], [492, 668], [493, 657], [497, 630
[[493, 731], [484, 699], [490, 708], [479, 672], [482, 690], [483, 679], [487, 652
[[486, 755], [477, 724], [483, 733], [472, 695], [476, 715], [476, 703], [480, 675
[[478, 772], [470, 740], [476, 749], [465, 712], [469, 731], [469, 720], [473, 692
[[472, 780], [463, 745], [470, 756], [458, 718], [462, 736], [463, 725], [466, 698
[[465, 774], [455, 741], [462, 751], [450, 714], [454, 732], [455, 721], [459, 694
[[459, 764], [449, 731], [456, 741], [444, 704], [448, 722], [449, 711], [453, 684
[[449, 747], [440, 714], [447, 724], [436, 686], [439, 705], [440, 693], [445, 666
[[441, 728], [433, 695], [439, 705], [428, 666], [432, 686], [433, 674], [437, 645
[[432, 708], [423, 675], [430, 685], [417, 645], [422, 666], [422, 654], [427, 624
[[424, 693], [415, 660], [422, 670], [409, 631], [413, 651], [414, 639], [418, 609
```

FACELANDMARK WITH GIVEN POINTS

```
import cv2
import mediapipe as mp
import time
from IPython.display import Video
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

class FaceMeshDetector:
    def __init__(self, staticMode=False, maxFaces=4, minDetectionCon=0.5, minTrackCon=0.5
        self.staticMode = staticMode
        self.maxFaces = maxFaces
```

```
            self.minDetectionCon = minDetectionCon
            self.minTrackCon = minTrackCon
            self.mpDraw = mp.solutions.drawing_utils
            self.mpFaceMesh = mp.solutions.face_mesh
            self.faceMesh = self.mpFaceMesh.FaceMesh(
                static_image_mode=self.staticMode,
                max_num_faces=self.maxFaces,
                min_detection_confidence=self.minDetectionCon,
                min_tracking_confidence=self.minTrackCon
            )
            self.drawSpec = self.mpDraw.DrawingSpec(thickness=1, circle_radius=2)

        def findFaceMesh(self, img, highlight_count=0):
            imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            results = self.faceMesh.process(imgRGB)
            faces = []
            if results.multi_face_landmarks:
                for faceLms in results.multi_face_landmarks:
                    face = []
                    for id, lm in enumerate(faceLms.landmark):
                        ih, iw, ic = img.shape
                        x, y = int(lm.x * iw), int(lm.y * ih)
                        face.append([x, y])

                        # Highlight only specified landmarks
                        if id < highlight_count:
                            cv2.circle(img, (x, y), 4, (0, 0, 255), -1)  # Highlight color (r
                    faces.append(face)
            return img, faces

    def main():
        # Get user input for number of landmarks to highlight
        try:
            highlight_count = int(input("Enter the number of landmarks to highlight : "))
            highlight_count = min(highlight_count, 468)  # Ensure it does not exceed 468
        except ValueError:
            print("Invalid input. Using default value of 68 landmarks.")
            highlight_count = 68  # Default value if input is invalid

        # Load your video file
        cap = cv2.VideoCapture("/content/drive/MyDrive/CV.mp4")  # Replace with the correct f
        pTime = 0
        detector = FaceMeshDetector(maxFaces=2)

        # Define video writer for output
        frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = cap.get(cv2.CAP_PROP_FPS)
        output_path = '/content/drive/MyDrive/output_with_given_points_landmarks.mp4'  # Spec
        out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (frame_width

        while cap.isOpened():
            success, img = cap.read()
            if not success:
                break
```

```
            img, faces = detector.findFaceMesh(img, highlight_count)
            if faces:
                print(faces[0])  # Print the first detected face landmarks for reference

            # Calculate FPS
            cTime = time.time()
            fps = 1 / (cTime - pTime) if (cTime - pTime) != 0 else 0
            pTime = cTime
            cv2.putText(img, f'FPS: {int(fps)}', (20, 70), cv2.FONT_HERSHEY_PLAIN,
                        3, (0, 255, 0), 3)

            # Write each frame to the output video
            out.write(img)

        cap.release()
        out.release()
        print(f"Video processing complete. Saved as {output_path}")

    # Run main to process the video
    main()

    # Display the output video
    Video("/content/drive/MyDrive/output_with_given_points_landmarks.mp4", embed=True)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
Enter the number of landmarks to highlight : 200
[[691, 305], [687, 280], [690, 289], [680, 258], [686, 273], [686, 263], [687, 242
[[692, 305], [686, 281], [690, 289], [681, 258], [686, 273], [686, 263], [687, 242
[[691, 306], [686, 281], [690, 290], [680, 258], [685, 273], [685, 264], [687, 242
[[691, 306], [685, 282], [689, 290], [679, 259], [684, 274], [684, 264], [686, 242
[[690, 306], [685, 282], [688, 290], [679, 259], [684, 274], [684, 264], [685, 242
[[691, 306], [685, 281], [689, 290], [679, 258], [684, 273], [684, 264], [686, 242
[[693, 305], [688, 280], [692, 289], [682, 257], [687, 273], [687, 263], [688, 241
[[694, 306], [689, 281], [692, 290], [682, 258], [688, 273], [688, 264], [689, 242
[[694, 307], [689, 282], [692, 291], [682, 259], [688, 274], [688, 265], [689, 243
[[694, 308], [689, 282], [692, 291], [682, 260], [688, 275], [688, 265], [689, 243
[[695, 308], [689, 283], [693, 292], [683, 260], [688, 275], [688, 265], [689, 244
[[697, 311], [691, 286], [694, 294], [684, 263], [690, 278], [690, 268], [691, 246
[[695, 311], [690, 286], [693, 295], [683, 263], [689, 278], [689, 269], [690, 247
[[695, 311], [689, 286], [693, 295], [683, 263], [688, 278], [688, 268], [689, 246
[[696, 311], [691, 287], [694, 295], [684, 263], [690, 279], [690, 269], [691, 247
[[695, 311], [690, 286], [693, 295], [683, 263], [689, 278], [689, 269], [690, 247
[[695, 309], [690, 284], [693, 293], [683, 261], [689, 276], [689, 266], [690, 244
[[696, 310], [691, 284], [694, 293], [684, 262], [690, 277], [689, 267], [690, 245
[[696, 311], [691, 286], [694, 295], [684, 263], [690, 278], [690, 269], [691, 247
[[695, 312], [690, 287], [693, 296], [683, 264], [689, 279], [689, 270], [690, 248
[[695, 313], [689, 288], [693, 296], [683, 265], [689, 280], [688, 270], [689, 248
[[695, 312], [690, 287], [693, 296], [683, 264], [689, 280], [688, 270], [689, 248
[[695, 311], [690, 287], [694, 295], [684, 263], [689, 279], [689, 269], [690, 247
[[696, 312], [691, 287], [694, 296], [684, 264], [690, 279], [690, 270], [691, 248
[[694, 312], [689, 287], [692, 296], [682, 264], [688, 280], [688, 270], [688, 248
[[694, 313], [689, 288], [692, 296], [682, 265], [688, 280], [688, 270], [689, 249
[[695, 315], [690, 290], [693, 298], [683, 267], [689, 282], [689, 273], [689, 251
[[694, 317], [689, 292], [692, 300], [683, 268], [688, 284], [688, 274], [689, 252
[[693, 317], [688, 292], [691, 301], [682, 269], [687, 284], [687, 275], [688, 253
[[695, 316], [690, 291], [693, 300], [684, 268], [689, 284], [689, 274], [691, 252
[[697, 317], [693, 292], [696, 300], [686, 268], [692, 284], [692, 274], [693, 252
[[698, 318], [693, 293], [696, 301], [687, 269], [693, 285], [692, 275], [693, 253
[[697, 318], [693, 293], [696, 302], [686, 270], [692, 285], [692, 275], [693, 253
[[697, 318], [693, 293], [696, 302], [687, 270], [692, 285], [692, 275], [693, 253
[[701, 319], [697, 294], [700, 302], [690, 270], [696, 286], [696, 276], [697, 254
[[703, 318], [699, 292], [701, 301], [692, 269], [698, 284], [698, 275], [699, 253
[[706, 316], [702, 290], [704, 299], [695, 267], [701, 282], [701, 272], [701, 251
[[709, 316], [704, 289], [707, 299], [697, 266], [704, 281], [703, 272], [704, 250
[[710, 315], [706, 288], [708, 297], [698, 265], [705, 280], [705, 271], [705, 249
[[711, 315], [706, 288], [709, 297], [699, 265], [705, 280], [705, 270], [705, 249
[[710, 314], [706, 288], [708, 297], [698, 265], [705, 280], [705, 270], [705, 249
[[710, 315], [705, 289], [708, 298], [698, 266], [705, 281], [704, 272], [705, 251
[[708, 316], [704, 290], [706, 299], [697, 267], [703, 282], [703, 273], [703, 252
[[705, 317], [700, 291], [703, 300], [694, 268], [699, 283], [699, 274], [700, 252
[[702, 317], [697, 291], [701, 299], [691, 268], [696, 283], [696, 273], [698, 251
[[698, 317], [692, 291], [696, 300], [685, 268], [690, 283], [691, 274], [692, 252
[[688, 318], [682, 292], [687, 301], [676, 269], [680, 284], [681, 275], [684, 252
[[678, 318], [670, 291], [676, 300], [666, 269], [669, 284], [670, 274], [674, 253
[[668, 318], [658, 291], [667, 300], [656, 270], [657, 284], [658, 275], [665, 254
[[657, 318], [645, 294], [655, 302], [644, 272], [644, 287], [646, 278], [654, 250
[[643, 320], [630, 294], [642, 303], [632, 273], [629, 287], [632, 278], [643, 257
[[631, 319], [618, 294], [631, 302], [622, 273], [617, 287], [621, 278], [633, 257
[[622, 318], [610, 294], [624, 302], [616, 273], [610, 287], [614, 278], [628, 258
[[615, 318], [604, 295], [617, 303], [610, 274], [603, 288], [607, 279], [622, 259
[[611, 317], [600, 294], [614, 302], [607, 274], [599, 287], [604, 279], [619, 259
[[612, 319], [599, 295], [614, 303], [607, 275], [599, 288], [604, 280], [619, 260
[[612, 319], [600, 295], [614, 303], [607, 275], [599, 288], [604, 280], [619, 260
[[612, 317], [600, 293], [615, 301], [608, 273], [600, 287], [605, 278], [620, 258
```