Python Script and Execution Result

Code from 'qr_scanner_web.py':

```
# ■■■ ■■■■■■ ■■■■■. import cv2 # OpenCV ■■■■■: ■■■ ■ ■■■ ■■■ ■■■■■.
import webbrowser # ■ ■■■■■ ■■■■ ■■■■■: QR ■■ ■■■ ■■■■ ■■■. import
time # ■■ ■■ ■■■■■: ■■ ■■ ■ ■■■■ ■■■■■. import numpy as np # NumPy
■■■■■: ■■ ■ ■■ ■■■ ■■■■■ ■■■■■. import re # ■■ ■■■(Regular
Expression) ■■■■■: URL ■■■■ ■■■■■. from PIL import ImageFont, ImageDraw,
Image # Pillow ■■■■■: ■■■■ ■■■ ■■ ■■ ■■■■■. # ■■ ■■■ ■■■■ ■■■
■■■■, ■■■ ■■ ■■■ ■■■■■. try: fontpath = "C:/Windows/Fonts/malgun.ttf" #
Windows■ ■■ ■■ ■■ ■■■■■. font = ImageFont.truetype(fontpath, 20) #
PIL(Pillow)■■ ■■■ ■■ ■■■ ■■■■■. is_font_loaded = True # ■■■ ■■■■■
■■■■■■ ■■■■ ■■■■■■. except IOError: is_font_loaded = False # ■■ ■■■
■■■■ ■■■■ False■ ■■■■■. font = None # ■■ ■■■ None■■ ■■■■■■. # ■■
■■■■ ■■■■ URL■ ■■■■ ■■■■■. # http ■■ https■ ■■■■ ■■■■ ■■■■.
url_pattern = re.compile(r'^(http|https)://[^\s/$.?#].[^\s]*$') def
put_text_on_frame(frame, text, pos, color=(255, 0, 0)): """■■ ■■■■ ■■■■
■■■■ ■■■■ ■■""" if is_font_loaded: # ■■ ■■■ ■■■■■■ ■■■■■. # OpenCV■
BGR ■■ ■■■ PIL■ RGB■ ■■■■■. frame_rgb = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB) # NumPy ■■■ PIL ■■■ ■■■ ■■■■■. pil_img =
Image.fromarray(frame_rgb) # ■■■■ ■■ ■ ■■ ■■■ ■■■■■. draw =
ImageDraw.Draw(pil_img) # ■■■ ■■■ ■■ ■■■■ ■■■■. draw.text(pos, text,
font=font, fill=color) # PIL ■■■■ ■■ OpenCV■ BGR ■■■■ ■■■■ ■■■■■.
return cv2.cvtColor(np.array(pil_img), cv2.COLOR_RGB2BGR) else: # ■■ ■■■
■■■■ ■■■ ■■ # ■■ OpenCV ■■■ ■■ ■■■■ ■■■■. cv2.putText(frame, text,
pos, cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2) return frame def
show_statistics(total_frames, recognized_frames, recognition_times): """ ■■
■■■ ■■■ ■■ ■■■■■. """ # ■■ ■■■ ■■■■■. recognition_rate =
(recognized_frames / total_frames) * 100 if total_frames > 0 else 0
avg_recognition_time = np.mean(recognition_times) if len(recognition_times)
> 0 else 0 # ■■ ■■■■ ■■■■■. stats_msg1 = f"■ ■■■ ■: {total_frames}"
stats_msg2 = f"QR ■■ ■■ ■■■ ■: {recognized_frames}" stats_msg3 = f"■■■:
{recognition_rate:.2f}%" stats_msg4 = f"■■ ■■ ■■:
{avg_recognition_time*1000:.2f}ms" # ■■ ■■■ ■■ ■■■ ■■■(■)■ ■■■■■.
height, width = 300, 500 stats_image = np.zeros((height, width, 3),
dtype=np.uint8) stats_image.fill(255) # ■■ ■■ # ■■ ■■■■ ■■■■ ■■■■■.
stats_image = put_text_on_frame(stats_image, "--- ■■ ■■ ---", (100, 50),
(0, 0, 0)) stats_image = put_text_on_frame(stats_image, stats_msg1, (50,
100), (0, 0, 0)) stats_image = put_text_on_frame(stats_image, stats_msg2,
(50, 140), (0, 0, 0)) stats_image = put_text_on_frame(stats_image,
stats_msg3, (50, 180), (0, 0, 0)) stats_image =
put_text_on_frame(stats_image, stats_msg4, (50, 220), (0, 0, 0))
cv2.imshow("Statistics", stats_image) cv2.waitKey(0) # ■■ ■■ ■■ ■■■ ■■
■■■■■. cv2.destroyAllWindows() def main(): """ ■■ ■■: ■■■ ■■ QR ■■■
■■■■ ■■■■ ■■■ ■■■. """ cap = cv2.VideoCapture(0) # 0■ ■■■(■■ ■■)■
■■■. if not cap.isOpened(): # ■■■ ■■■ ■■■■ ■■■■■. print("■■■ ■ ■
■■■■.") return detector = cv2.QRCodeDetector() # QR ■■ ■■ ■ ■■■■ ■■ ■■■
■■■■■. print("QR ■■ ■■■ ■■ ('q'■ ■■ ■■)") last_data = None # ■■■■■
■■■ ■■■■ ■■■■ ■■ ■■■ ■■■■■. last_open_time = 0 # ■■■■■ ■■■ ■ ■■■
■■■■■. DELAY_TIME = 5 # ■■ QR ■■■ ■■ ■■■■■ ■■ ■■ ■■(■)■ ■■■■■.
total_frames = 0 # ■ ■■■ ■■ ■■■■■. recognized_frames = 0 # QR ■■■ ■■■
```

```
■■■ ■■ ■■■■■. recognition_times = [] # ■■■ ■■ ■■■ ■■■■■. while True: #
■■ ■■■ ■■■■ ■■■■■ ■■■■ ■■■■■. ret, frame = cap.read() # ■■■■ ■ ■■■■
■■■■■. if not ret: # ■■■■ ■■ ■ ■■■■ ■■■ ■■■■■. break total_frames += 1
start_time = time.time() # ■■ ■■ ■■■ ■■■■■. display_frame = frame.copy()
found_data = False data = None points = None # 1. ■■ ■■■■■ ■■ ■■ data,
points, _ = detector.detectAndDecode(frame) if data: found_data = True # 2.
■■■■ ■■ ■■ ■, ■■■ ■■■■ ■■ ■■ ■■ ■■ if not found_data: # ■■■■ ■■■■
■■■■ ■■■ ■■■■. gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
enhanced_frame = cv2.equalizeHist(gray_frame) data, points, _ =
detector.detectAndDecode(enhanced_frame) if data: found_data = True # ■■■
■■ ■■ ■■ ■ ■■ ■■ ■■ display_msg = "QR ■■■ ■■ ■■■■..." color = (255, 0,
0) # ■■■ if found_data: end_time = time.time() # ■■ ■■ ■■■ ■■■■■.
recognition_times.append(end_time - start_time) recognized_frames += 1
is_same_qr = (data == last_data) is_delay_passed = (time.time() -
last_open_time) > DELAY_TIME # URL ■■■ ■■ is_valid_url =
url_pattern.match(data) if is_valid_url and (not is_same_qr or
is_delay_passed): display_msg = "QR ■■ ■■ ■■! ■■■ ■■■." color = (0, 255,
0) # ■■■ print(f"QR ■■ ■■ ■■: {data}") try: webbrowser.open(data) except
Exception as e: print(f"URL ■■ ■■: {e}") last_data = data last_open_time =
time.time() elif is_valid_url: display_msg = f"■■ ■■:
{data}\n■■■({DELAY_TIME}s) ■■ ■■ ■..." color = (0, 255, 255) # ■■■ else:
display_msg = f"■■■ URL■ ■■■■: {data}" color = (0, 0, 255) # ■■■ # QR ■■
■■ ■■ ■■■ if points is not None and len(points) > 0: points =
np.int32(points).reshape(-1, 2) cv2.polylines(display_frame, [points], True,
color, 3) # ■■■■ ■■■ ■■ display_frame = put_text_on_frame(display_frame,
display_msg, (10, 30), color) # ■■■ ■■ ■■ ■■ cv2.imshow("QR Code Scanner",
display_frame) # 'q' ■■ ■■■ ■■■■ ■■ if cv2.waitKey(1) & 0xFF == ord('q'):
break # CPU ■■■■ ■■■ ■■■ ■■■ ■■■ ■■■■ ■■ 1■ ■■■■ ■■■■■.
time.sleep(0.01) # ■■ ■■ ■ ■■ ■■ cap.release() cv2.destroyAllWindows()
show_statistics(total_frames, recognized_frames, recognition_times) if
__name__ == "__main__": main()
```

Execution Output:

```
QR ■ ■ ('q' ) QR ■ ν : http://en.m.wikipedia.org QR ■ ν :
http://en.m.wikipedia.org
```