AI Vision Security Guard: Automated CSV Log Engine in PictoBlox

Introduction

Ever wanted an intelligent security system that doesn't just sit there recording blank footage? This project introduces an automated security logger built entirely in **PictoBlox Python**.
Using computer vision via the **Face Detection** extension, the system monitors a room. The moment a human face is detected, it triggers a clean **File Automation pipeline** that dynamically checks your computer's operating system, creates a dated folder for that day, and appends a precise time-stamped entry inside an Excel-compatible .csv log sheet. By building a latching lock into the logic, the system prevents log spamming while someone remains in frame.

Step 1: Materials and Software Prerequisites

You don't need any expensive external hardware components to get this up and running—just your computer!

- **Software:** PictoBlox IDE (with Python interface support)
- **Hardware:** Built-in Laptop Webcam or external USB Webcam
- **PictoBlox Extension: Face Detection** (Make sure to add this via the extensions menu at the bottom-left of the PictoBlox interface so the underlying machine learning models load into memory).

Step 2: The Core Python Architecture

We structured this project using professional **Object-Oriented Programming (OOP)**. Instead of messy global code blocks, wrapping variables and functions inside a clean class keeps file system tracking separate from live camera frames.
Copy and paste this clean code into your PictoBlox Python editor script window:

```
Python
import
import

class IntelligentSecurityLogger:
    def __init__
        """Initializes the file automation paths and the AI Face Detection module."""
        # 1. Automated File Directory Setup
                                f"Security_Logs_{time.strftime('%Y_%m_%d')}"
                            f"{self.folder_name}/activity_log.csv"


        # 2. PictoBlox AI Camera Setup

                        "on"  0        # Camera on, 0% transparency
                                       # Draw bounding box around faces
                              0.5      # 50% confidence threshold
```

```
        "AI Camera Initialized. Scanning room..."

    # 3. State Tracking Variable (Prevents duplicating logs)
                                    False

def _setup_file_system
    """Internal helper method to handle folder and CSV creation."""
    if not

            f"Created folder: {self.folder_name}"

    if not
        with open                   "w"  as
                    "Timestamp,Event_Type,Status\n"
            "Initialized new CSV log file."

def log_event
    """Appends a new security alert line into the CSV file."""
                                '%Y-%m-%d %H:%M:%S'
    with open                  "a"  as
                f"{timestamp},Human Detected,ALERT\n"

def monitor
    """Main execution loop that keeps the security scanner running."""
    try
        while True
                                    # Process current camera frame

            # Check if a face is in the frame
            if                  0
                if not
                                                '%H:%M:%S'
                        f"[{current_time}] Alert: Human detected in frame!"

                                    # Trigger file automation log
                                            True

            else
                # Reset the logging latch when the room becomes empty
                if
                        "Room is clear again. Resetting monitor."
                                            False

                    0.2   # Maintain stable CPU performance

    except
```

```
                "\nSecurity System safely deactivated."

# Execution Block
if                  "__main__"
```

Step 3: Understanding How the Code Works

Let's look at how the code organizes your data pipeline under the hood:

1. **File Automation (_setup_file_system):** It fetches the current calendar date from your system time clock and formats it (Year_Month_Day). Using the os library, it checks if that specific daily directory exists. If not, it builds it automatically. This gives you self-organizing storage.
2. **AI Counting Engine (fd.count()):** PictoBlox handles complex computer vision backends behind the scenes. We use fd.analysecamera() to slice the camera stream into frame matrices, and fd.count() checks if any human face vectors cross our 50% confidence barrier.
3. **The Latching Mechanism (self.person_already_logged):** If someone stands in front of the camera for 10 seconds, a raw loop would write to the spreadsheet 50 times. Our logic uses a boolean state flag to "lock" writing after the first frame. It safely unlocks only when the room goes completely clear again.

Step 4: Testing Your New Intelligent Logger

To run your project and see it work:

1. Open PictoBlox, choose the **Python Environment**, and ensure the **Face Detection** module is loaded.
2. Click the green **Run** button. Your camera feed will open up.
3. Step away from the camera's viewpoint initially. Then step into the frame—the console will flash an immediate alert!
4. Stop the script and open the directory on your hard drive where PictoBlox saves script assets. Open up your daily Security_Logs folder, and launch activity_log.csv. You will find your precise time-stamped alerts recorded in clean rows!

💡 **Pro Tip for your Instructables Post:** Don't forget to take a couple of screenshots or record a quick 5-second video snippet of your computer webcam highlighting a face bounding box and the generated CSV rows. Adding those visuals directly into your Instructables steps will grab a lot of attention!