CodyNick in Python Demo Session

Demo projects to present CodyNick in python, including AI and IoT tools.

- Presenter Guide

Presenter Guide

CodyNick Fabric Presenter Guide: Build a CodyNick Temperature Alarm

Session goal

Build a temperature alarm in eight small steps. The audience first sees how easily Python can control a physical LED, then gradually adds blinking, joystick input, a display, a sensor, temperature colors, an alarm melody, and joystick alarm control. Then turn the same sensor into a live browser dashboard, give the program eyes and voice control, and finish with an offline AI conversation. There are fourteen steps in the full demonstration; pause between them to let the audience predict what comes next.

These first examples intentionally use the simplest possible code. They do not include connection checks, error handling, or cleanup structures. The goal is to make the first experience with CodyNick clear and approachable.

Do not paste the final program at the beginning. Type each step live, run it, invite the audience to predict the result, and then change the existing program for the next step.

Equipment

- Raspberry Pi running CodyNick
- CodyJoy Pro
- Temperature sensor
- Stand-alone seven-segment display
- Python IDE at `http://10.42.0.1/code/`
- Dashboard in a second browser tab at `http://10.42.0.1/dashboard/`
- USB camera for the camera steps; connect it when that step begins
- USB microphone and speaker for the voice steps

Prepare before the audience arrives

- Use CodyNick 0.9.8 or later for the tested temperature dashboard example.
- Check the temperature sensor, seven-segment display, CodyJoy Pro LEDs, microphone, and speaker before the session. The hardware examples at the beginning are deliberately short.
- Open the IDE and dashboard in separate tabs. If projecting, arrange the windows side by side for the dashboard reveal so the audience can see both code and results.
- Rehearse `local_ai_voice_conversation.py` once. At presentation time, still let its three models finish loading before inviting the audience to speak.
- Generate the prerecorded answers for the saved-answer conversation before the session, as described in that step. The Local AI conversation does not use that generator.

Presentation rhythm

For every step:

1. Ask the engaging question before showing the new code.
2. Let the audience suggest an answer.
3. Type the small change.
4. Ask the audience to predict the result.
5. Select **Run This File** and observe the hardware and live terminal.

Step 1: Light One RGB LED

Ask before starting

> How many lines of Python do you think we need to control a real LED?

Explain that `CodyNick.CN()` creates the connection and LED number `0` is the first LED on the RGB matrix. A color is written as red, green, and blue values in hexadecimal.

```
import time
import CodyNick


cody = CodyNick.CN()
```

```
CodyNick.RGB_Matrix.set(cody, 0, "#00FF00")
time.sleep(3)

CodyNick.RGB_Matrix.clear(cody)
```

Point out

- `0` selects the first LED.
- `#00FF00` means green.
- `time.sleep(3)` keeps it visible for three seconds.
- `clear` turns the LED off at the end.

Step 2: Make the LED Blink

Ask before starting

> A computer follows instructions very quickly. How can we make it wait long enough for our eyes to see an LED blink?

Replace the single LED command and three-second wait with this loop:

```
import time
import CodyNick

cody = CodyNick.CN()


for count in range(5):
    CodyNick.RGB_Matrix.set(cody, 0, "#00FF00")
    time.sleep(0.5)
    CodyNick.RGB_Matrix.set(cody, 0, "#000000")
    time.sleep(0.5)
```

Point out

- `range(5)` repeats the indented instructions five times.
- `#000000` means no red, green, or blue, so the LED is off.
- The two half-second waits create an obvious rhythm.

Invite the audience to predict what changing `0.5` to `0.1` will do.

Step 3: Control LEDs with the Joystick

Ask before starting

> Can one physical control choose which light Python turns on?

Move the joystick up, right, down, and left. One LED in the corresponding part of the matrix lights up in a different color. Press the joystick to light the center LED white.

```
import time
import CodyNick

cody = CodyNick.CN()

while True:
    direction = CodyNick.Joystick.states(cody, "CJP")
    CodyNick.RGB_Matrix.clear(cody)

    if "CLICK" in direction:
        CodyNick.RGB_Matrix.set(cody, 6, "#FFFFFF")
    elif "UP" in direction:
        CodyNick.RGB_Matrix.set(cody, 13, "#0000FF")
    elif "RIGHT" in direction:
        CodyNick.RGB_Matrix.set(cody, 4, "#00FF00")
    elif "DOWN" in direction:
        CodyNick.RGB_Matrix.set(cody, 1, "#FF0000")
    elif "LEFT" in direction:
        CodyNick.RGB_Matrix.set(cody, 8, "#FFFF00")

    time.sleep(0.1)
```

Point out

- `states` reads the current joystick direction.
- Up uses LED 13 in blue.
- Right uses LED 4 in green.
- Down uses LED 1 in red.
- Left uses LED 8 in yellow.
- Clicking uses LED 6 in white.
- `if` and `elif` choose only one LED at a time.
- The matrix is cleared before showing the current direction.
- `while True` keeps reading the joystick until the program is stopped.

Step 4: Show a Fixed Number

Ask before starting

> If Python can control one light, can it also send a complete number to a display with one instruction?

Replace the blinking code with a seven-segment display command:

```
import CodyNick


cody = CodyNick.CN()


CodyNick.Seven_Segment.display(cody, "Stand-Alone", 25)
```

Point out

- `"Stand-Alone"` selects the separate seven-segment display.
- `25` is the value sent to the display.

Step 5: Show a Counter

Ask before starting

> The display is showing data from our program. What should change if we want the display to show progress instead of one fixed value?

Change the fixed display command to a loop:

```
import time
import CodyNick

cody = CodyNick.CN()



for number in range(1, 11):
    CodyNick.Seven_Segment.display(cody, "Stand-Alone", number)
    print("Count:", number)
    time.sleep(1)
```

Point out

- `range(1, 11)` produces the numbers `1` through `10`.
- The same variable is shown on the physical display and in the live terminal.
- Software variables can directly represent real-world output.

Step 6: Display the Measured Temperature

Ask before starting

> So far, Python has displayed numbers that we chose. How can the physical world choose the number instead?

Replace the counter with a continuous sensor loop:

```
import time
import CodyNick

cody = CodyNick.CN()



while True:
    temperature = CodyNick.Temperature_Sensor.read(cody)
    temperature = round(temperature, 1)

    CodyNick.Seven_Segment.display(cody, "Stand-Alone", temperature)
    print("Temperature:", temperature, "C")

    time.sleep(1)
```

Point out

- `read(cody)` gets a current measurement from the sensor.
- `round(..., 1)` keeps one digit after the decimal point.
- `while True` continues until the presenter stops the script.
- The one-second pause gives a readable update rate.

Ask someone to gently warm the sensor and watch the number respond.

Step 7: Add Temperature Colors

Ask before starting

> Can someone understand whether a temperature is cold, comfortable, or hot more quickly from a number or from a color?

Add the temperature decision and use a loop to color all 16 LEDs:

```
import time
import CodyNick
```

```
cody = CodyNick.CN()



while True:
    temperature = CodyNick.Temperature_Sensor.read(cody)
    temperature = round(temperature, 1)

    CodyNick.Seven_Segment.display(cody, "Stand-Alone", temperature)

    if temperature < 20:
        color = "#0000FF"
    elif temperature <= 30:
        color = "#00FF00"
    else:
        color = "#FF0000"

    for led in range(16):
        CodyNick.RGB_Matrix.set(cody, led, color)

    print("Temperature:", temperature, "C")
    time.sleep(1)
```

Point out

- Below 20 C is blue.
- From 20 C through 30 C is green.
- Above 30 C is red.
- `if`, `elif`, and `else` turn one measurement into one of three decisions.
- `range(16)` applies the chosen color to the complete matrix.

Step 8: Add the Alarm and Joystick Inhibit

Ask before starting

> A real alarm may need to be silenced while someone investigates. How can one joystick button act as both the silence control and the re-enable control?

Start with two variables that remember the alarm state:

```
alarm_enabled = True
was_clicked = False
```

Then build the final program:

```
import time
import CodyNick

cody = CodyNick.CN()

alarm_enabled = True
was_clicked = False


while True:
    temperature = CodyNick.Temperature_Sensor.read(cody)
    temperature = round(temperature, 1)

    CodyNick.Seven_Segment.display(cody, "Stand-Alone", temperature)

    if temperature < 20:
        color = "#0000FF"
    elif temperature <= 30:
        color = "#00FF00"
    else:
        color = "#FF0000"

    for led in range(16):
        CodyNick.RGB_Matrix.set(cody, led, color)

    is_clicked = CodyNick.Joystick.click(cody, "CJP")

    if is_clicked and not was_clicked:
        alarm_enabled = not alarm_enabled
```

```
        if alarm_enabled:
            print("Alarm enabled")
        else:
            print("Alarm inhibited")

    was_clicked = is_clicked

    if temperature > 30 and alarm_enabled:
        CodyNick.CJP_Sound_Maker.play_until_done(cody, "C6", 120)
        CodyNick.CJP_Sound_Maker.play_until_done(cody, "E6", 120)
        CodyNick.CJP_Sound_Maker.play_until_done(cody, "G6", 120)
        CodyNick.CJP_Sound_Maker.play_until_done(cody, "E6", 120)

    print("Temperature:", temperature, "C")

    time.sleep(0.3)
```

Point out

- `alarm_enabled` remembers whether sound is permitted.
- `was_clicked` allows one physical press to cause only one toggle.
- `not alarm_enabled` changes enabled to inhibited and inhibited to enabled.
- The red visual warning remains active while the audible alarm is inhibited.
- The alarm melody is `C6`, `E6`, `G6`, `E6`.
- The joystick is checked between complete melody cycles.

Step 9: Turn the Temperature into a Live Dashboard

Ask before starting

> The sensor can change a light and sound an alarm. Could the same few lines of Python also create a live measuring instrument in a browser?

Keep CodyJoy Pro and the temperature sensor connected. Keep the dashboard tab hidden until the program runs, then reveal the gauge and graph together.

In the IDE, open **CodyNick Examples -> dashboard -> `temperature_on_dashboard.py`**. Select **Run This File**, then open `http://10.42.0.1/dashboard/` in the second tab. Running this example clears existing dashboard cards and creates its own two cards.

For a live coding presentation, use the complete tested example below. First draw attention to the two `Dashboard.Card(...)` calls, then to the two `.set(...)` updates. The familiar sensor-reading loop is still the center of the program.

```
import time
import CodyNick
import Dashboard

cody = CodyNick.CN()

Dashboard.clear()

temperature_card = Dashboard.Card(
    "gauge",
    "Temperature",
    color="green",
    min_value=0,
    max_value=50,
    value=0,
    font_size=32,
    font_name="Arial",
    horizontal_alignment="center",
    vertical_alignment="center"
)

history_card = Dashboard.Card(
    "graph",
    "Temperature History",
    color="blue",
    value=[],
    column_span=3,
    x_axis_label="Time (s)",
    y_axis_label="Temperature (°C)"
)
```

```
history = []
Cnt = 0
started = time.monotonic()

while True:
    temperature = CodyNick.Temperature_Sensor.read(cody)
    Cnt += 1

    if temperature is not None:
        temperature = round(temperature, 1)

        if temperature != 0 and temperature < 120:
            print(f"{Cnt} : {temperature}")

            temperature_card.set(temperature)

            seconds = round(time.monotonic() - started, 1)
            history.append([seconds, temperature])
            history = history[-20:]
            history_card.set(history)

    time.sleep(2.5)
```

The reveal

1. Let two or three valid readings arrive. The graph needs at least two points.
2. Show the current temperature on the gauge and its history on the graph.
3. Ask a volunteer to gently warm the sensor with their hand. Allow time for the probe to respond; avoid touching or wetting exposed electronics.
4. Point to the changing value, then follow the rising line across the graph.
5. Ask: "If the room cools again, what should happen to this line?" Let the audience predict it.
6. Leave the chart running briefly so the audience sees the history rather than just one number.

Point out

- `import Dashboard` gives Python access to the on-device dashboard.
- `Dashboard.Card(...)` creates a widget; `.set(...)` sends it a new value.

- `column_span=3` gives the history graph three dashboard columns on a wide screen.
- Each point is `[elapsed seconds, temperature]`. The X-axis is time since this run began, not a sample counter or a clock time.
- The latest 20 valid readings are retained. Their time values continue increasing when earlier points leave the graph.
- `Cnt` counts read attempts in the terminal. It is not used for the graph's X-axis.
- Missing readings are checked before rounding. This demonstration also skips zero and values at or above 120 C. This is a demo filter, not a general sensor-validity rule: a genuine zero-degree reading would also be skipped, and negative readings are accepted.
- Rejected readings leave both cards unchanged and create a time gap before the next point.
- The gauge begins at zero until the first accepted value. Its needle is configured for 0 to 50 C; use comfortable room-temperature changes for this demonstration.
- The script attempts another reading after a 2.5-second pause; sensor and database work add to that interval, so the plotted timestamps reflect actual elapsed time.

Transition to AI

> We can measure and visualize something the program can sense. Now, what if its next input is a picture rather than a number?

Switch back to the IDE and replace the running program with the next example.

Step 10: Connect to AI and Take a Picture

Ask before starting

> We have given our program sensor inputs. What new things could it understand if we gave it eyes?

The camera has not been connected during the earlier steps. Connect the USB camera now and wait a few seconds for the Raspberry Pi to recognize it.

Replace the previous program with this short camera program:

```
import CodyNick
from codynick_ai import CodyNickAI

cody = CodyNick.CN()
ai = CodyNickAI(workspace="/home/client", camera_index=0)

picture = ai.take_picture(
    "camera_demo",
    cody=cody,
    get_ready_sound=True
)

print("Picture saved:", picture)

ai.close()
cody.close()
```

Run the program, then open **Images** in the IDE and select `camera_demo.jpg` to show the captured picture to the audience. Running the program again replaces the same picture instead of filling the Images folder with many files.

Point out

- `CodyNickAI` connects the student program to the installed AI tools.
- `cody=cody` connects the camera activity to CodyJoy Pro.
- `get_ready_sound=True` plays the get-ready and shutter sounds.
- `take_picture()` captures one image and releases the camera after the shot.
- `"camera_demo"` is the reusable image name.
- The image is stored in `/home/client/images` and appears under **Images** in the IDE.
- Taking a picture is the first step. The same picture can next be used for object detection, OCR, face tools, or another AI model.

Step 11: Detect Objects in the Picture

Ask before starting

> The camera can take a picture, but can the computer tell us what is inside it?

Place a few familiar objects in front of the camera, such as a bottle, cup, chair, or person. Extend the camera program so it captures a fresh picture and analyzes it:

```
import CodyNick
from codynick_ai import CodyNickAI

cody = CodyNick.CN()
ai = CodyNickAI(workspace="/home/client", camera_index=0)

picture = ai.take_picture(
    "camera_demo",
    cody=cody,
    get_ready_sound=True
)

print("Picture saved:", picture)

ai.load_app("yolo", model="nano")
result = ai.detect_objects("camera_demo")

print("Objects found:", len(result["detections"]))

for item in result["detections"]:
    print(item["class_name"], item["confidence"])

ai.close()
cody.close()
```

After the result appears, open **Images**, open the `results` folder, and select `camera_demo_objects.jpg`. Show how the AI has drawn a box and label around each recognized object.

Point out

- `load_app("yolo", model="nano")` loads the fast object-detection model.
- `detect_objects("camera_demo")` analyzes the picture captured by the program.
- `result["detections"]` contains the objects found by the model.
- Each result includes a class name and a confidence score.
- The annotated image is saved under **Images → results**.
- The model may miss an object or label it incorrectly. Better lighting and a clear view usually improve the result.

Step 12: Control the LEDs with Your Voice

Ask before starting

> We have controlled the lights with code and a joystick. Can we control them without touching the computer or the gadget at all?

Use a USB microphone, or use the microphone built into the USB camera if it has one. Keep it close enough to hear the presenter clearly.

In the IDE, open **CodyNick Examples →** `voice_led_colors.py` and select **Run This File**. Wait until the terminal says that offline English speech recognition is ready, then say one command at a time:

- `red`
- `green`
- `blue`
- `yellow`
- `white`
- `purple`
- `lights off`
- `stop listening`

The installed example includes useful progress messages. This shorter presenter version keeps the same commands, recognition settings, hardware behavior, and cleanup logic:

```
import CodyNick
from codynick_ai import CodyNickAI



COMMAND_COLORS = {
    "red": "#FF0000",
    "green": "#00FF00",
    "blue": "#0000FF",
    "yellow": "#FFFF00",
    "white": "#FFFFFF",
    "purple": "#FF00FF",
```

```
}

COMMANDS = list(COMMAND_COLORS) + ["lights off", "stop listening"]


def fill_matrix(cody, color):
    for led in range(16):
        CodyNick.RGB_Matrix.set(cody, led, color)


def main():
    cody = CodyNick.CN()
    ai = CodyNickAI(workspace="/home/client")
    listener = None

    try:
        if not cody.ensure_connected():
            raise RuntimeError("CodyNick gadget not found.")

        ai.load_stt(model="small", language="en")

        listener = ai.listen(
            commands=COMMANDS,
            min_confidence=0.45,
            device="auto"
        )

        for event in listener:
            if not event.get("accepted"):
                continue

            command = event["matched_command"]
            print("Heard:", command)

            if command == "stop listening":
                break
            elif command == "lights off":
                CodyNick.RGB_Matrix.clear(cody)
            else:
```

```
                fill_matrix(cody, COMMAND_COLORS[command])

    finally:
        if listener is not None:
            listener.stop()

        ai.close()
        CodyNick.RGB_Matrix.clear(cody)
        cody.close()


main()
```

Point out

- `load_stt(...)` loads offline speech-to-text before listening starts. The spoken audio does not need to be sent to an internet service.
- `COMMANDS` limits recognition to the phrases used by this demonstration.
- `device="auto"` selects an available USB or camera microphone.
- `min_confidence=0.45` rejects uncertain matches below the selected confidence level.
- `lights off` clears the LEDs but keeps the microphone listening.
- `stop listening` ends the program and turns the LEDs off.
- The `finally` section releases the microphone and clears the hardware even when the program is stopped unexpectedly.

Demonstration tips

- Wait for the ready message before speaking.
- Speak one command clearly, then pause briefly for the result.
- Reduce nearby conversation and music while demonstrating recognition.
- If a command is rejected, move closer to the microphone and repeat it normally.

Step 13: Hold a Voice Conversation

Ask before starting

> The system can recognize one spoken command. Could it stay awake, understand several different questions, and answer us with a different response each time?

Watch the completed demonstration before presenting this step:

Your browser cannot play this video. Use the download link below.

Open or download the voice conversation demo video

Connect a USB microphone and speaker. Keep CodyJoy Pro connected for the LED and buzzer feedback.

Prepare the answers once

In the IDE, open **CodyNick Examples →** `generate_conversation_answers.py` and select **Run This File**. Wait until all 27 messages have been generated. They are stored in:

```
/home/client/audio/conversation_answers
```

The answers are generated in advance so the live conversation can respond immediately. Running the generator again safely builds a complete replacement set before replacing the old files. It does not delete unrelated recordings.

Start the live conversation

Open **CodyNick Examples →** `voice_conversation.py` and select **Run This File**. Wait for `Ready. Say: wake up`, then demonstrate this sequence:

1. Say `wake up` or `please wake up`.
2. Wait for the ready sound and a spoken response such as `I am listening`.
3. Ask several questions without saying the wake phrase again.
4. Say `sleep` or `go to sleep` to return immediately to wake mode.
5. Say `wake up` again and continue the conversation.
6. Stop speaking for ten seconds and show that it returns to wake mode automatically.

Try questions from these groups:

Topic	Example phrases
Greeting	`hello`, `hi`, `good morning`, `good afternoon`, `good evening`, `good night`
Identity	`what is your name`, `who are you`, `tell me about yourself`

Topic	Example phrases
Well-being	`how are you`, `how are you today`, `are you okay`
Capabilities	`what can you do`, `how can you help me`, `what do you do`
Favorite color	`what is your favorite color`, `which color do you like`
Feelings	`are you happy`, `how do you feel`, `are you sad`
Goodbye	`goodbye`, `bye`, `see you later`

Watch the gadget emotions

State	CodyJoy Pro feedback
Waiting	One dim blue LED
Wake phrase accepted	Green LEDs and two rising notes
Listening	Four cyan center LEDs; buzzer remains silent
Processing	Yellow side LEDs
Answering	Green pattern and a short opening note
Question not understood	Two orange flashes and descending notes
Goodbye	Purple LEDs and a farewell melody

Point out

- The speech recognition and generated answers work offline after installation.
- Each topic has three prerecorded answers, selected without immediately repeating the previous answer.
- Unique playback names prevent the audio cache from replaying an earlier response.
- The microphone stops while the buzzer or speaker is active, so the program does not recognize its own sounds.
- The program remains awake across several questions and sleeps only after a command, goodbye, or ten seconds of silence.
- Pressing **Stop** releases the microphone, speech worker, serial connection, and LEDs.

Demonstration tips

- Run the answer generator before the audience arrives; it loads the large TTS model and is intentionally slower than normal playback.
- Keep the microphone away from the speaker and reduce nearby conversation.
- Pause after the wake phrase so the ready response can finish before asking a question.
- Ask one short supported question at a time and wait for the answer.

- If a question is not recognized, repeat it clearly using one of the listed phrases.

Step 14: Let the Audience Talk with Local AI

Ask before starting

> Until now, we chose the commands and prepared the answers. What if someone asks a simple question that we never wrote into the program?

Connect CodyJoy Pro, a USB microphone, and a speaker. Open **CodyNick Examples → `local_ai_voice_conversation.py`** and select **Run This File**.

The installed filename is `local_ai_voice_conversation.py`; there is no separate `local_ai_conversation.py` in the current release. Use the installed example so its model loading, microphone handling, and gadget feedback have already been prepared. No answer-generation script is needed for this version.

Set up the finale

Tell the audience: "The device is loading its ears, its voice, and its language model." Wait for these terminal messages before speaking:

```
Text-to-speech is ready.
Speech recognition is ready.
Local AI is ready. Say: Wake up
```

Loading happens once at the start of each run. Each new answer still needs time for language generation and speech synthesis. Let the LED states show that work to the audience.

Demonstrate, then hand over to the audience

1. Say `wake up`. Wait for the rising notes, green lights, and "I am listening."
2. Ask `What is an RGB LED?` and wait for the spoken answer.
3. Ask `Name one use for a temperature sensor.` without saying the wake phrase again. This brings the story back to the dashboard and alarm the audience just watched.
4. Invite a volunteer to ask a different simple question, such as `What is a joystick?` or `Why do plants need water?`. Explain that the answer is generated, so its wording can vary.

5. Say `sleep` and wait for "Going to sleep." Show that ordinary questions no longer start answers until `wake up` is spoken again.
6. Wake it once more. Say `stop listening` or `goodbye` for the closing melody and cleanup.

If the volunteer pauses for ten seconds while the program is waiting for a question, it returns to wake mode. Say `wake up` again and let the ready response finish.

Watch the conversation travel through the system

Part	What the presenter can point to
Speech-to-text (STT)	The recognized question appears after `You:` in the terminal
Local language model	A generated answer appears after `AI:`
Text-to-speech (TTS)	The speaker reads that answer aloud
CodyJoy Pro	Blue waiting LED, green wake effect, cyan listening LEDs, yellow processing LEDs, green answering pattern, purple farewell

Point out

- STT, TTS, and the language model work locally after installation. Internet is not required for the conversation itself.
- All three models load once before the first wake phrase and are reused until the script stops. They are released when it ends.
- Answers contain at most nine space-separated words. Short answers help the pace, but the language model can still make mistakes; ask the audience to judge the answer.
- The microphone is stopped during gadget sounds, speech synthesis, and playback, helping prevent the program from hearing its own answer. Wait for cyan listening LEDs.
- The ready response includes `;;;` before its text, as do generated answers sent to TTS. This is the prepared playback pause used in the tested script.
- Each generated reply replaces `llm_answer.wav` in the device's audio folder. It does not create a growing folder of answer recordings.
- Unlike Step 13's prepared topics, this example accepts open questions. Each question is independent: it does not remember earlier answers or read the temperature sensor.

Presenter recovery tips

- If the recognized question after `You:` is wrong, repeat it clearly rather than treating the generated reply as a model failure.

- If no ready message appears, wait for model loading or inspect the terminal error before inviting an audience question.
- Keep the speaker away from the microphone and leave pauses between questions.
- Use a complete question each time, such as "What is a joystick?" rather than "What about that?", because this example has no conversation memory.

Closing audience questions

- Which thresholds would you use for a refrigerator, greenhouse, or classroom?
- What other sensor could replace the temperature sensor?
- What could the joystick control besides the alarm?
- What could we measure on a second dashboard card?
- Why is elapsed time more useful than a sample counter on a temperature graph?
- What could an AI model discover in the picture we just captured?
- How could the program react when it recognizes a particular object?
- What other spoken commands could control a CodyNick project?
- What new questions and emotions would you add to the conversation?
- How is a generated answer different from one we recorded in advance?

Presenter reminder

Keep the code visible while demonstrating the hardware response. The purpose is not to memorize every command; it is to show the repeated pattern:

1. Read an input.
2. Make a decision.
3. Control an output.
4. Repeat safely.