Reachy Mini Robot Academy

Phase 1 · Understand the mechanism. Write the code. Test your idea.

Contents

MISSION 0 · 25 MINUTES

Motor Detective

Solve the "Couldn't start daemon" mystery with clues, careful tests, and safe help from an adult.

Clue first, then a safe next step!

Today you will make…

A detective report that separates a network clue from a motor clue.

Start here. You need the prepared course folder, Reachy Mini Control, and a notebook. No repair tools.

You are ready to move on when you can:

- Read the exact status without moving Reachy.
- Explain why a missing motor is a clue, not proof of a loose cable.

Words to discover

evidence **motor** **debugging** **safety**

The daemon mystery

Reachy Mini shows a generic "Couldn't start daemon" screen. That sounds spooky, but detectives do not guess: we collect clues. Our path is Observe → Read the real error → Check one boundary → Test one idea → Verify. Today's 25-minute plan: 4 minutes to explore, 4 to predict, 10 to build and test, 5 to write your discoveries, and 2 to collect your badge. Bonus challenges are optional.

Nine servos, three jobs

A servo is a motor system that aims for a target position and checks where it actually is. Reachy Mini has nine: six cooperate to position its head, one turns its body, and two turn its antennas. The head is a parallel mechanism: several rods support the same platform, like a team holding up one tray.

Two kinds of connection do different jobs. Servo arms and rods carry force to the head. Electrical cables carry power and data between electronics and servos. The daemon is the helper program talking to this motor system. Wi-Fi reaches the robot computer; the internal motor bus reaches the servos.

A sticker number, a software name, and an electronic ID are different labels. In the pinned SDK configuration, stewart_1 has ID 11—not ID 1. Copy the full error exactly; never change IDs as a child's troubleshooting step.

Reachy Mini SDK 1.10.0 · electronic addresses, not wiring order

Job	Software name	ID
Body rotation	body_rotation	10
Six head servos	stewart_1 … stewart_6	11 … 16
Right antenna	right_antenna	17
Left antenna	left_antenna	18

Read the real robot status

First open Terminal: Command-Space, type Terminal, press Return. Type cd and a space, drag the prepared course folder from Finder into Terminal, then press Return. Run source .venv/bin/activate to use the toolbox your adult installed. If that fails, ask for setup help; do not install anything yourself.

Run the command below from that folder. This status reader only reads a message; it does not command a motor. Copy the exact daemon clue, including names such as stewart_1. A summary is not the full error.

Optional after Mission 1: save these three lines in practice.py and run python3 practice.py. This invented clue stays on your Mac.

Tiny Python practice · runs on your Mac

```
clue = "stewart_1 did not respond"
print("My pretend clue:", clue)
print("A clue is not proof of a loose wire.")
```

```
python3 examples/mission_00/check_robot.py
```

Complete runnable file in the companion folder: examples/mission_00/check_robot.py. The web edition includes the full source and a Copy source button.

Hint: In Terminal, type pwd and ls. The course folder should contain examples. Do not type the $ prompt. Mission 1 shows how to save your own Python file.

Predict the kind of clue

Question: Reachy Mini answers on Wi-Fi, but the screen says “Missing motors.” What is the best prediction?

Wi-Fi is the problem because the robot answered.

Answer: The robot is reachable, but its motor backend needs attention.

Press buttons until the message disappears.

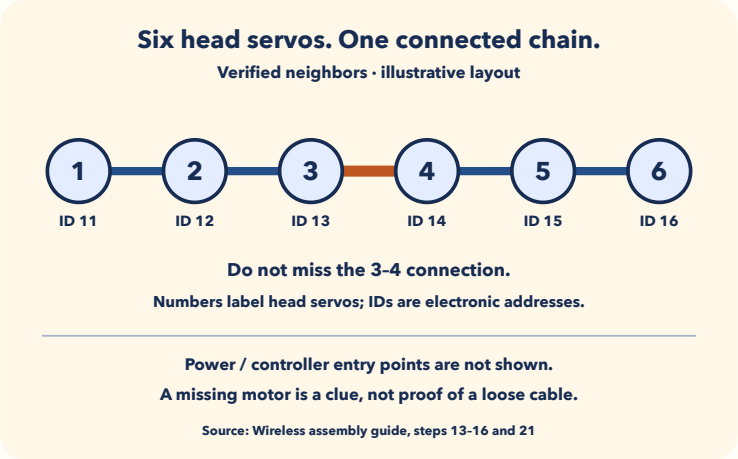
Trace a connection, not a guess

The manufacturer’s Wireless assembly guide connects head motors 1–2, 2–3, 3–4, 4–5, and 5–6. That 3–4 cable matters: this is not two separate three-motor branches. Cable routing and controller connections must be checked against the guide for your robot revision.

A report of only motor 1 missing makes its link to motor 2 one possible place for an adult to inspect. A cable, connector, motor fault, or motor setup could cause it; the message cannot prove which. Groups of missing motors can suggest a shared path problem, but only after you know the actual wiring and where it is powered. Do not open or repair wiring: save the clue and get an adult.

Paper lab • trace the path

Use the verified neighbor chain 1–2–3–4–5–6, but pretend the ONLY power and data entry is at 6. This entry point is a learning assumption, not verified wiring. Cover 2–1: motor 1 loses the path. Cover 4–3: 3, 2, and 1 lose it. A real missing response is not proof of a cable fault.



Read the words beside this diagram too; they are the instructions.

Hint: With the lab's pretend source at 6, a break between 4 and 3 affects 3, 2, and 1. A break between 2 and 1 affects only 1. Real entry points must be checked against the correct build guide.

Observe before touching

Read the status once and copy its exact wording. Robot not found is a network or discovery clue. Healthy-but-relaxed means the daemon can be ready while joints are resting. A motor fault means stop motor commands and get an adult; do not repair or wake a faulty robot. You can still complete today's browser experiment without hardware.

Read an error, then test one idea

Error card — One motor: Stop and get an adult. Power off before connection checks; an adult can inspect that motor's connections and setup. Error card — Several motors: Stop and get an adult. A shared path is one hypothesis, not a proven branch fault; use the correct wiring guide. Error card — All motors: Stop and get an adult. Power off before connection checks; an adult can inspect motor power and communication.

STOP / ADULT / CHECK — Stop and get an adult

Hint: Write the exact motor name or ID, not just "motor broken." Add whether the robot answers on the network. Leave cables connected.

Keep a detective notebook

Record one clue: network, one motor, several motors, all motors, or daemon ready. Write "I think ___ because ___," then one safe next step. A ready daemon does not prove torque is enabled or the robot is physically healthy. Leave the extra challenges for another day.

Motor Detective badge

You earned it by observing, reading the real error, checking one boundary, testing one idea, and verifying what changed. A great detective protects the robot and asks for help at the right time.

Explain it back

Only one motor is missing. What have you proved?

▼ **Explain it first, then compare**

Only that this motor did not respond as expected. A cable is one hypothesis; power, the motor, or its configuration could also be involved.

Take this skill forward: Every robot needs power and communication. Check these as separate questions.

Want another experiment? Find Mission 0 in the optional challenge bank after the nine missions.

Adult reading

- Manufacturer's Wireless assembly guide (steps 13–16 and 21)
- Pinned SDK 1.10.0 motor names and IDs

MISSION 1 · 25 MINUTES

Hello, Reachy!

Use Terminal and a Python file to greet Reachy, then read a safe status value.

Hello from a tiny Python programmer!

Today you will make…

Your first Python greeting, then an antenna-position reading without a movement request.

From Mission 0: a working network is not the same as healthy motors.

You are ready to move on when you can:

- Save a plain-text .py file and run it from Terminal.
- Change a variable and predict the new printed greeting.

Words to discover

Terminal **file** **print** **string** **variable** **import** **with** **position reading**

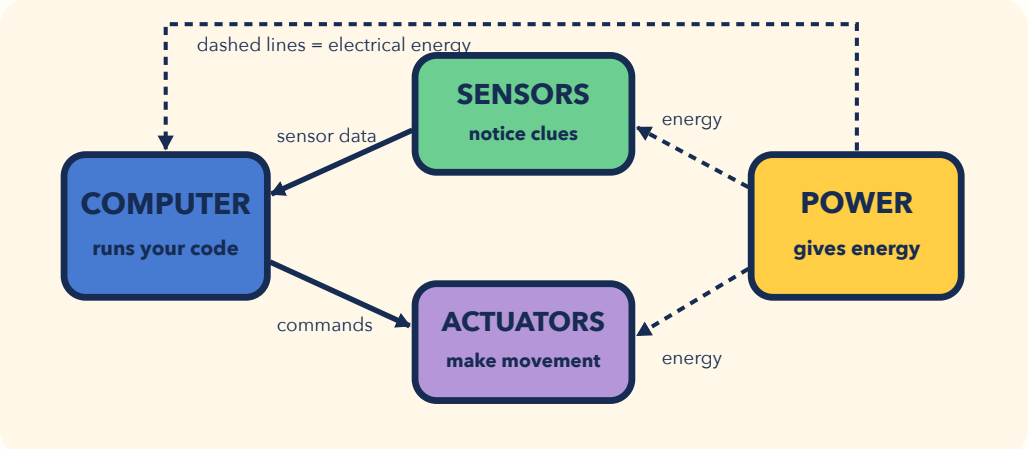
Say hello with code

Make a Python file and run it from the course folder. We will print words on the Mac, then read antenna positions without requesting movement. Today's 25-minute plan: 4 minutes to explore, 4 to predict, 10 to build and test, 5 to write your discoveries, and 2 to collect your badge. Bonus challenges are optional.

Reading and moving are different requests

A robot has a computer that runs code, sensors that collect information, actuators that make physical actions, and a power supply. Today's example reads antenna positions; it sends no movement target.

No movement request is not the same as a completely read-only connection. The SDK can configure a control setting when it connects. Use the adult-prepared robot with other control apps inactive, and keep the movement area clear. Mission 0 is the strictly read-only status check.



Read the words beside this diagram too; they are the instructions.

Print a greeting

Use `print` to show words. Text in quotes is a string; robot_name is the variable holding it. An import brings a helper into the file. The full example uses `with open_robot() as mini:` to open and later close an SDK connection; `with` does not make code read-only.

First make a tiny program of your own. In TextEdit choose Format → Make Plain Text, then turn off Edit → Substitutions → Smart Quotes. Type these three lines exactly; Python needs straight quotes. Save as practice.py in the course folder, keeping the .py ending (choose "Use .py" if asked). This small exercise only prints on your Mac.

Tiny Python practice · runs on your Mac

```
robot_name = "Reachy"
print("Hello,", robot_name)
print("I am learning Python!")
```

```
python3 examples/mission_01/hello_reachy.py
```

Complete runnable file in the companion folder: examples/mission_01/hello_reachy.py. The web edition includes the full source and a Copy source button.

Hint: Use straight quotes: robot_name = "Reachy". The name on the left is the variable; the words inside quotes are its value. Save before running again.

Predict the screen

The program prints a greeting, reads antenna positions, and never calls a motion method. What should happen? Make your prediction before running it.

Reachy moves its head.

Answer: Python prints a greeting and antenna positions.

The program changes a motor.

Build a name tag

Open the course folder in Terminal as in Mission 0. In each new Terminal window, run source .venv/bin/activate first. Then run python3 practice.py. Expect "Hello, Reachy" and "I am learning Python!" Change only Reachy inside the quotes, save, and run again. Finally, run the prepared robot command above from the same folder.

Hint: If your file is practice.py.txt, change it to practice.py in Finder. If Python cannot find it, check the Terminal folder with pwd.

Read the position result

Find "Position check succeeded" followed by two antenna positions in radians, ordered [right, left] from Reachy's viewpoint. Your numbers may differ. This shows a reading succeeded; it does not prove all motors are healthy or that motion is safe.

Find a tiny typo

If Python says it cannot open the file, check that you are in the course folder and the file is practice.py, not practice.py.txt. SyntaxError often means a missing quote or parenthesis; NameError often means a misspelled variable. Read the last error line, change one thing, save, and retry. Never paste the Terminal prompt or printed answer into the file.

> **Hint:** SyntaxError means Python could not read your code. Start with the line number and check quotes and parentheses; it does not mean a motor is broken.

Record your first program

Write your variable name, the command you ran, and where the greeting appeared. Explain how reading a position differs from asking for movement.

Earn the Hello badge

You can save, run, and change a Python greeting—and distinguish a position reading from a movement request.

Explain it back

Does print("Move!") move Reachy? Why?

▼ **Explain it first, then compare**

No. print displays text on the Mac. A robot action requires a command to the robot SDK.

Take this skill forward: Variables, strings, and print work even when you change robots.

Want another experiment? Find Mission 1 in the optional challenge bank after the nine missions.

MISSION 2 · 25 MINUTES

Reachy Moves Carefully

Discover how a servo corrects its position, then change the timing of a small real greeting.

Small, safe moves are super moves!

Today you will make…

A slower, small greeting—and an explanation of the servo loop behind it.

From Mission 1: a variable stores a value. Today a time value changes a real gesture.

You are ready to move on when you can:

- Explain target → measure → compare → correct.
- Distinguish a head pose from an individual servo angle.
- Change only duration and compare the same greeting twice.

Words to discover

servo **feedback** **target** **pose** **parameter** **duration**

The motor that checks its work

Imagine pointing at a spot with your eyes closed. How would you know you reached it? A position servo has a sensor to answer that question. Today: 6 minutes exploring feedback, 4 on Python, 3 predicting and rehearsing, 8 comparing the real gesture, and 4 explaining your result. The coordinate picture and extra challenges can wait for another day. Keep fingers, hair, clothes, and objects outside the movement area.

Inside a servo: aim, measure, correct

A motor makes rotation. Gears trade speed for turning force, called torque. The output shaft and horn pass that turning force to a linkage. A position sensor measures the shaft angle; a controller compares that measurement with a target and drives a correction. This repeating check is feedback. It is control, not human-like thinking.

Position error = target angle − measured angle. If the error shrinks, the shaft is getting closer. Reaching the target does not mean the servo is off: it may need effort to hold against gravity. Never test this by holding a moving robot or blocking a joint.

Reachy's six head servos do not each control one head direction. They move six linked arms and rods together to support and position one platform. Python asks for a head pose; the SDK calculates the coordinated joint targets. Do not send guessed angles to individual motors.

Paper lab · be the controller

Draw a servo shaft and a target arrow. Target = 20°, measured = 0°. In this model, move halfway toward the target: 0 → 10 → 15. Compute the error after each step: 20, 10, 5. What changes if the target is −20°? Without a position reading, the error is unknown. This is a teaching rule, not the real controller.

One variable, one controlled change

A function is a named action; a parameter is a setting you give it. `create_head_pose(roll=5, degrees=True)` describes a small head tilt in degrees. `goto_target()` requests a movement toward a pose. In this example only the antenna values use `np.deg2rad` to change degrees into radians, another angle unit. `duration` sets time in seconds; `body_yaw=None` leaves the body target unspecified.

Open practice.py from Mission 1, replace its contents with these four lines, save, then run python3 practice.py. This practice prints numbers only. Next, find MOTION_DURATION near the top of the prepared greeting file; the program passes it into each of three movement calls. You do not need to understand the connection and error-handling helpers yet.

Tiny Python practice · runs on your Mac

```
seconds = 1.5
print("Original move:", seconds, "seconds")
seconds = 2.0
print("Slower move:", seconds, "seconds")
```

```
python3 examples/mission_02/greeting_move.py
```

Complete runnable file in the companion folder: examples/mission_02/greeting_move.py. The web edition includes the full source and a Copy source button.

Hint: MOTION_DURATION is seconds for each target, not for the whole program. Three moves at 2.0 seconds take about 6 seconds, plus connection and program overhead.

Sketch the motion first

Why does the example use body_yaw=None? Make your prediction, then draw the neutral and tilted poses before running.

Answer: Leave the current body yaw unchanged by this request.

Make the motion instant.

Convert degrees to radians.

Rehearse, then edit one line

Try the browser rehearsal first. Keep the same three poses and compare 1.5 with 2.0 seconds per target. Predict the total planned movement time. Longer duration means a lower average speed for the same path; it does not make a larger movement safe.

Then clear the real movement area and use the adult-approved setup. Run the prepared greeting once. After a normal result, open examples/mission_02/greeting_move.py. Change only MOTION_DURATION = 1.5 to MOTION_DURATION = 2.0, save, and run once more. Leave all angles and other lines unchanged. If the robot is unavailable, record the browser result and mark the real test “not run.”

Paper lab • rehearse the greeting

Draw three frames: neutral → small tilt → neutral. At 1.5 seconds per target the planned motion lasts 4.5 seconds; at 2.0 it lasts 6.0. Connection time is extra. The SDK coordinates six head motors; a 5° head roll does not mean each servo turns 5°.

STOP / ADULT / CHECK – Use the adult-approved setup and clear the movement area before running.

Hint: Rehearse in the browser first. In greeting_move.py, edit only MOTION_DURATION = 1.5 to MOTION_DURATION = 2.0. Do not copy the browser feedback model into a hardware controller.

Collect a fair comparison

Watch from a safe distance. Both versions request neutral → small tilt → neutral, with body_yaw=None. Compare 4.5 seconds of planned motion with 6.0 seconds; connection and program time are extra. Write: "I changed time. I kept the poses the same. I observed ___." Stop for red lights, unusual noise, heat, or stuck movement.

Stop for a warning

Red stop card: stop the program and get an adult for red lights, unusual noise, heat, binding, or a stuck head. Never force a joint by hand. If the move is not what you predicted, do not assume the pose is safe or known; have an adult check Reachy before retrying.

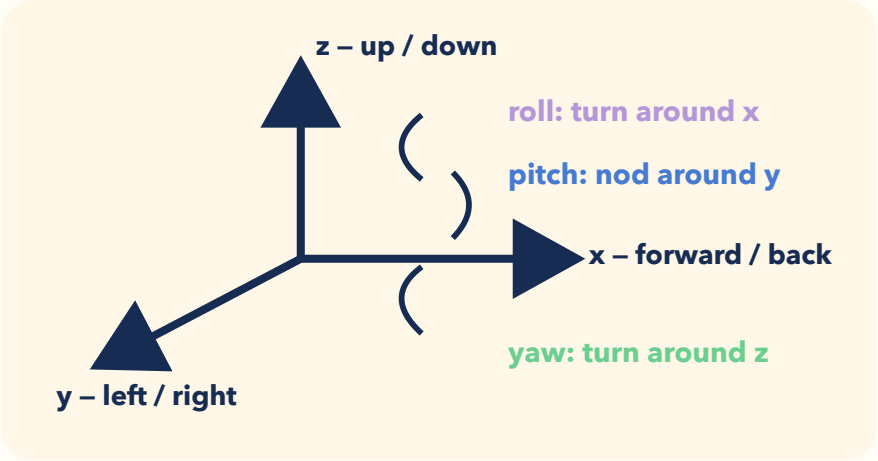
STOP / ADULT / CHECK – Stop and get an adult for red lights, noise, heat, binding, or stuck movement.

Hint: If Python reports a spelling error, compare your edited line with the original. For unexpected physical motion, use the stop procedure and get an adult instead of retrying.

Record it; then explore coordinates if you have time

Record both durations, whether the gesture looked the same, and whether it took longer. Explain what the servo sensor measures and what Python asks the head to do.

Optional deeper look: in Reachy's frame, x points forward, y left, and z up. Roll turns around x, pitch around y, and yaw around z. A pose combines position and orientation. These head coordinates are not the nine electronic motor IDs.



Read the words beside this diagram too; they are the instructions.

Earn the Careful Mover badge

You explained feedback and compared one timing change. Mark whether your evidence came from the browser rehearsal or the real robot.

Explain it back

A servo wants 20° but measures 5°. What is its error? Does a 5° head roll mean every servo turns 5°?

▼ **Explain it first, then compare**

The error is 20 − 5 = 15°. No: the SDK calculates coordinated joint targets for the head pose. Servo angles and head angles describe different things.

Take this skill forward: Feedback and careful timing will matter for MicroDuck joints and a robot-arm gripper too.

Want another experiment? Find Mission 2 in the optional challenge bank after the nine missions.

Adult reading

- Reachy Mini hardware: head mechanism and nine motors
- ROBOTIS XL330 position-servo reference

MISSION 3 · 25 MINUTES

Reachy Sees Pixels

Capture one local BGR camera frame and learn why sensing pixels is not understanding.

Pixels are tiny clues made of light!

Today you will make…

A local toy photograph with its pixel dimensions explained.

A servo has a position sensor inside. A camera is a different sensor: it measures light.

You are ready to move on when you can:

- Identify height, width, and color channels in an image shape.
- Change only the scene lighting and compare what you see.

Words to discover

camera frame **pixel** **sensor** **BGR** **shape** **privacy**

A camera notices light

Reachy's camera is a sensor. It collects one picture, called a frame, as tiny colored squares called pixels. Today's 25-minute plan: 4 minutes to explore, 4 to predict, 10 to build and test, 5 to write your discoveries, and 2 to collect your badge. Bonus challenges are optional.

Sensing is not understanding

A frame is raw sensor data. OpenCV commonly gives a color image as a BGR array with shape (height, width, 3): blue, green, red values for each pixel. Seeing those numbers is not the same as understanding what a scene means.

Inspect one safe frame

Run the prepared camera example from the repository root. It retries for up to five seconds, prints the NumPy shape, and saves a local JPEG. It never uploads an image. If WebRTC is unavailable, inspect the supplied still at `examples/assets/practice_frame.ppm` instead.

Replace practice.py with these four lines and run python3 practice.py. The star means multiply. Predict the pixel count before running; you should get 307200. A camera image with this height and width has that many pixels, with three color numbers per pixel.

Tiny Python practice · runs on your Mac

```
height = 480
width = 640
pixels = height * width
print("Pixel count:", pixels)
```

```
python3 examples/mission_03/take_picture.py
```

Complete runnable file in the companion folder: examples/mission_03/take_picture.py. The web edition includes the full source and a Copy source button.

Hint: The * symbol multiplies. Height × width counts pixels; the final 3 in the shape counts color channels, not extra pictures.

Predict the shape

Predict which number in a shape like (480, 640, 3) is height, which is width, and why the last number is 3.

Frame a non-private scene

Place a toy in the camera's existing view; do not turn the head by hand. Keep faces, private papers, passwords, and screens out of the frame. Using the supplied still is practice, not a new camera capture.

Hint: Use a toy on plain paper. Before capture, check the entire view for faces, names, screens, and private papers.

Capture one local frame

Run the command once. Read the shape, then run open examples/mission_03/reachy_snapshot.jpg in Terminal to view the saved JPEG. The file stays on this Mac and is replaced on the next capture; rename a copy in Finder before comparing two pictures. Do not upload it.

Check the view

If no frame arrives, ask an adult to check the daemon and WebRTC stream, or use the supplied practice still. If the picture is dark, change the light or toy position—not cables or motor angles. A camera failure is not permission to open Reachy.

Hint: Use the supplied still if the stream fails. Label it "practice image," not "live camera." Never claim a capture worked just because the program printed a filename.

Explain the pixels

Record the BGR shape and one bright pixel area. Write why a camera frame is data, not understanding, and why this mission keeps the image local.

Earn the Pixel Explorer badge

You can describe pixels, BGR shape, sensing, understanding, and local camera privacy.

Explain it back

A frame is (480, 640, 3). Does Reachy now know which toy it sees?

▼ **Explain it first, then compare**

No. It has 480 rows, 640 columns, and 3 color values per pixel. Recognizing the toy needs additional processing and can be wrong.

Want another experiment? Find Mission 3 in the optional challenge bank after the nine missions.

MISSION 4 · 25 MINUTES

Reachy Speaks

Play a short local sound and learn how speaker output is made from waveform samples.

Kind words, clear voice!

Today you will make…

One short greeting that you actually hear, not just a playback message.

Camera = input. Speaker = output. Which one collects information?

You are ready to move on when you can:

- Explain why a speaker is an output.
- Separate “play requested” from “I heard it.”

Words to discover

output **speaker** **waveform** **sample** **recorded sound** **generated speech** **verification**

Sound is robot output

A speaker is an output device. It turns a stream of digital samples into moving air that people hear. We will play one short, kind greeting. Today’s 25-minute plan: 4 minutes to explore, 4 to predict, 10 to build and test, 5 to write your discoveries, and 2 to collect your badge. Bonus challenges are optional.

Waveforms are sample lists

A waveform is a changing signal. Each sample is a number for the signal at one instant. A recorded sound stores samples captured by a microphone; generated speech is made by a text-to-speech system. They are different ways to create audio.

Ask the speaker to play

Run the prepared example from the repository root. It asks `mini.media.play_sound()` to play `examples/assets/hello_reachy.wav`, waits for its non-blocking playback window, and uses the WebRTC media backend. The WAV is synthetic local eSpeak speech, not a human recording or cloud-generated audio, so the offline lesson still works without a robot.

Replace practice.py with these three lines and run python3 practice.py. Square brackets hold a list. len counts its items, and [0] selects the first item: Python starts counting positions at zero. These five pretend samples explain the idea; they do not make sound.

Tiny Python practice · runs on your Mac

```
samples = [0, 1, 0, -1, 0]
print("Sample count:", len(samples))
print("First sample:", samples[0])
```

```
python3 examples/mission_04/speak.py
```

Complete runnable file in the companion folder: examples/mission_04/speak.py. The web edition includes the full source and a Copy source button.

Hint: Square brackets make a list. len(samples) counts its items; samples[0] reads the first. The practice list explains samples but makes no sound.

Predict the waveform

Predict what a waveform graph looks like during silence and during a louder part. Predict why playback needs a wait after `play_sound()`.

Choose considerate output

Preview the supplied WAV on the Mac: run open examples/assets/hello_reachy.wav in Terminal. This is synthetic speech, not a person inside Reachy. Use the volume your adult checked and tell nearby listeners before playback. Never surprise someone with a loud sound.

Hint: Start with the supplied clip at the adult-checked volume. Listen once from a comfortable distance; do not increase volume repeatedly to solve a connection problem.

Listen for the result

At the prepared setup, run the exact command once. The program reports that it asked Reachy to play; use your ears as the observation and ask a listener what they heard. The sound file travels from this Mac to Reachy over your local network. It is not sent to cloud AI.

Check sound and connection

If no sound plays, first listen to the WAV on the computer. If the computer plays it but Reachy does not, ask an adult to check the robot's sound setup. Do not raise the volume repeatedly or open the robot.

Hint: A successful request is only one clue. Does the WAV exist? Was playback requested? Was sound heard? Record the first missing step.

Compare two sound sources

Write one difference between a recorded sound and generated speech. Sketch a waveform sample and note whether your listener heard the intended greeting.

Earn the Clear Speaker badge

You can identify an output device, explain samples and waveforms, and check whether a listener actually heard your greeting.

Explain it back

The program says "play requested," but the room is silent. Was the test successful?

▼ **Explain it first, then compare**

Not yet. The request was sent, but sound was not verified. Check the playback path with an adult rather than claiming Reachy spoke.

Want another experiment? Find Mission 4 in the optional challenge bank after the nine missions.

MISSION 5 · 25 MINUTES

Reachy Listens Carefully

Record one explicit three-second clip and learn why speech recognition is a fallible guess.

Listening for three careful seconds!

Today you will make…

A deliberate three-second recording, with an optional checked transcript.

A camera frame does not automatically mean understanding. The same is true of recorded sound.

You are ready to move on when you can:

- Record only after everyone nearby agrees.
- Find one place where a transcript could be mistaken.

Words to discover

input **microphone** **sample rate** **noise** **recognition** **fallibility** **privacy**

A microphone is an input sensor

A microphone turns changing air pressure into samples. Our program connects first, then counts down. Wait for “Recording now” to speak; “Recording stopped” marks the end. The target window is three seconds, but chunks and device delays can change the saved duration. Today’s 25-minute plan: 4 minutes to explore, 4 to predict, 10 to build and test, 5 to write your discoveries, and 2 to collect your badge. Cloud transcription is optional and adult-supervised.

Rate, noise, and guesses

Sample rate says how many samples are measured each second. Reachy’s input is commonly 16,000 samples per second. Background noise, quiet speech, accents, and overlapping voices can all make recognition wrong. A transcript is a guess, not a fact.

Record one bounded clip

Run the prepared example from the repository root. It visibly counts down, records one bounded three-second window through `mini.media`, and saves `examples/mission_05/recorded_audio.wav`. A device chunk can straddle the window edge, so the program reports the captured sample duration honestly. It does not continuously listen. A supplied WAV from Mission 4 remains an offline fallback for practicing the file steps without a microphone.

Replace practice.py with these four lines and run python3 practice.py. You should see 48000 samples for a pretend three-second, 16000-samples-per-second recording. The actual recording helper reads the device rate and reports the duration it really captured.

Tiny Python practice · runs on your Mac

```
sample_rate = 16000
seconds = 3
sample_count = sample_rate * seconds
print("Expected samples:", sample_count)
```

```
python3 examples/mission_05/listen.py
```

Complete runnable file in the companion folder: examples/mission_05/listen.py. The web edition includes the full source and a Copy source button.

Hint: Samples per second × seconds = sample count, per channel. This multiplication predicts an ideal clip; check the real saved duration separately.

Predict the noisy word

Predict which safe practice word could be confused in a noisy room, and name one sound that could cause the confusion.

Choose safe practice words

Use an invented greeting. Ask everyone nearby and wait for agreement before running. Do not record names, addresses, passwords, private conversations, or anyone who has not agreed. A child's voice can itself be personal data, even when the words are invented.

Hint: Choose an invented greeting and speak only between the recording messages. Even an invented phrase spoken by a child needs the adult's under-13 privacy check before upload.

Record, then listen locally

Run once and read the actual captured duration. Leave the privacy prompt waiting; in Finder, open examples → mission_05 → recorded_audio.wav and listen. The next recording replaces this file. Type no unless your adult is present and has approved both the clip and the under-13 safeguards in the appendix. Only yes/y permits transcription. A failed upload does not prove the data stayed local.

Treat recognition as fallible

If recognition is wrong, compare the audio with the transcript and try a quieter room or slower safe phrase. Do not repeat a cloud upload automatically. If the microphone or daemon fails, stop and ask an adult to check it.

Hint: Listen locally before trusting a transcript. Bad recording and wrong transcription are different problems. Do not keep uploading the same clip to guess which one failed.

Use the red adult checkpoint

Write the sample rate and actual duration printed by the program, one noise source, and whether you skipped transcription. If used, compare its guessed words with the clip. Speech recognition can also use AI; its job here is to turn sound into text, not invent an answer. Mission 7 explores generating a reply.

STOP / ADULT / CHECK – Stop before uploading a child's voice: an adult must check under-13 data safeguards, privacy, and cost. Local recording can skip the cloud.

Earn the Careful Listener badge

You can explain a short recording window, check its actual duration, and treat recognition as a guess. Record whether transcription was used or skipped.

Explain it back

Why check the transcript before using it to choose an action?

▼ **Explain it first, then compare**

It is a model's guess about the audio, not a perfect record. A wrong word can lead to the wrong choice.

Want another experiment? Find Mission 5 in the optional challenge bank after the nine missions.

MISSION 6 · 25 MINUTES

Reachy Decides with Rules

Use visible if, elif, and else rules to choose a safe named gesture without AI.

If this, then that—let's check the rule!

Today you will make…

A predictable local decision rule with a safe fallback.

Inputs are clues; outputs are actions. A rule connects them.

You are ready to move on when you can:

- Trace an if/else choice without running it.
- Test a matching word, a non-matching word, and an empty input.

Words to discover

condition **Boolean** **if** **elif** **else** **deterministic rule** **fallback**

These choices use rules, not a learned model

Reachy can choose from a small menu using ordinary Python. With the same input and code, a deterministic rule makes the same choice; different inputs can follow different branches. This example does not learn or call cloud AI. Some AI systems also use rules, so "has rules" alone does not tell us whether something is AI. Today's 25-minute plan: 4 minutes to explore, 4 to predict, 10 to build and test, 5 to write your discoveries, and 2 to collect your badge.

Conditions are Boolean questions

A condition is a question with a true or false answer. Python calls those Boolean values. `if` checks the first condition, `elif` checks another only when the earlier one was false, and `else` is the safe fallback for everything unknown.

Run the local decision program

Run this offline example from the repository root. Type `hello`, `dance`, `sleep`, or a made-up word. It prints a text response and one named gesture identifier; no network, cloud model, `eval`, generated code, or motor angle is involved.

Replace practice.py with this code. Copy the four spaces before each print; they tell Python which lines belong to a branch. Run python3 practice.py three times, changing only word to "hello", "sleep", then "banana". The double equals asks whether two values match; a single equals stores a value.

Tiny Python practice · runs on your Mac

```
word = "hello"
if word == "hello":
    print("Hello, friend!")
elif word == "sleep":
    print("Good night!")
else:
    print("Try hello or sleep.")
```

```
python3 examples/mission_06/decide.py
```

Complete runnable file in the companion folder: examples/mission_06/decide.py. The web edition includes the full source and a Copy source button.

Hint: The colon starts a block. Indent the lines inside it by four spaces. The == operator asks “are these equal?”; = assigns a value.

Trace every branch

Before running, predict the response for `hello`, `dance`, `sleep`, and `maybe`. Which input reaches `else`? Explain why only one branch responds.

Change a greeting, keep the rule

First test all three branches of practice.py. Then open examples/mission_06/decide.py and change only “Hello! Reachy says hi.” to another kind greeting. Save, run the mission command, and type hello. The response text changes; the gesture name stays WAVE. This file prints names only. Movement comes later through a fixed local menu, never guessed angles.

Hint: Write expected results for three inputs before testing. Unknown input should reach the prepared fallback, not invent a new movement.

Test all four paths

Run the exact command once for each practice word. Check that every input gets exactly one text response and that unknown input gets the fallback. This test is fully offline and does not move Reachy.

Check order and spelling

If your new branch never runs, print or inspect the cleaned word and check the earlier conditions. A rule can be predictable and still be wrong if the input spelling or branch order is wrong.

Hint: Print the input and selected action name in a local practice program. Check capitalization and spaces before blaming hardware.

Explain the decision

Write one input, its Boolean question, its branch (`if`, `elif`, or `else`), and its named gesture. Contrast this transparent rule with a model prediction from Mission 7.

Earn the Rule Builder badge

You can explain conditions and Booleans, trace if/elif/else, and keep unknown inputs inside a safe fallback.

Explain it back

Why give an unknown command a fallback instead of guessing a motor angle?

▼ **Explain it first, then compare**

A known fallback keeps the possible actions bounded. Unrecognized text is not a safe physical target.

Want another experiment? Find Mission 6 in the optional challenge bank after the nine missions.

MISSION 7 · 25 MINUTES

Reachy Meets Cloud AI with Care

Ask one short, non-private question and learn how APIs and model predictions need adult guardrails.

Explore the cloud with grown-up guardrails!

Today you will make…

One short cloud conversation with a clearly drawn privacy boundary.

A local rule follows your conditions. A cloud model predicts a response and can be wrong.

You are ready to move on when you can:

- Point to exactly what data leaves the Mac.
- Check the reply and keep it separate from robot control.

Words to discover

cloud **API** **model** **prediction** **secret** **privacy** **cost** **latency** **inaccuracy**

A cloud helper has boundaries

A cloud service runs on computers somewhere else. An API is the doorway a program uses to request that service. This is an adult checkpoint: an API key is a secret, and each request can have privacy, account, and cost consequences. Today's 25-minute plan: 4 minutes to explore, 4 to predict, 10 to build and test, 5 to write your discoveries, and 2 to collect your badge. Bonus challenges are optional.

Models predict patterns

A model predicts likely text from patterns. It is not a person, a database of truth, or a deterministic if/elif rule. It can be useful, late, inaccurate, biased, or inappropriate, so people verify important answers and ask an adult when unsure.

Ask one safe question

Run this one-call example from the repository root. An adult must install the approved requirements and set `OPENAI_API_KEY` in the environment; the learner must never paste it into code. The program warns about names, addresses, school, passwords, personal information, and private conversations, then asks exactly one non-private question.

Replace practice.py with these three lines and run python3 practice.py. Change only the words inside the quotes to make another imaginary robot question. This practice stays on your Mac. len counts characters, including spaces; keep the result below 500 before using the cloud helper.

Tiny Python practice · runs on your Mac

```
question = "Invent a name for a moon robot."
print("My question:", question)
print("Character count:", len(question))
```

```
python3 examples/mission_07/ask_cloud_ai.py
```

Complete runnable file in the companion folder: examples/mission_07/ask_cloud_ai.py. The web edition includes the full source and a Copy source button.

Hint: Practice with the local snippet first. API keys are credentials, not words to add to a Python lesson or mission log.

Predict before you ask

Predict one made-up robot question the model may answer helpfully and one answer you would verify elsewhere. Predict how a slow network (latency) might change what the learner sees.

Constrain the request

With your adult present, try one invented robot-story question of at most 500 characters. The helper asks for two short sentences and a local guard limits displayed text to 280 characters. Each request has a 20-second timeout with no retries; this is not a precise whole-program timer. A short reply can still be wrong or unkind. If that happens, stop and tell your adult. Without approved cloud setup, write an imagined reply on paper and mark the cloud test “not run.”

Hint: Choose an invented, non-private question. If an adult has not enabled cloud access, do the paper input → possible reply exercise and mark the cloud test “not run.”

Review one prediction

With an adult, run the exact command once and read the answer as a suggestion. The helper uses the Responses API with `store=False`, which asks that response not be stored for later API retrieval; it is not a promise of zero retention. Check the account’s current privacy and data controls.

Read a cloud error without guessing

Missing setup needs an adult. A timeout means the request did not finish in time; it may still have sent data or incurred cost. A rate-limit response can mean too many requests or an account quota problem, not just a busy service. An incomplete or empty response is reported as a failure, not invented robot speech. Do not retry repeatedly. The length guard is not a content filter or a truth check.

Hint: A timeout is not permission to click repeatedly. Stop after the bounded request and preserve the non-private error for an adult.

Record the cloud safety gate

Record a non-private prompt and one claim to verify. Ask your adult which model is configured: the prepared default is gpt-5-mini, but an environment setting can override it. If you did not use the cloud, say so. Never record a key or claim that store=False means nothing is retained.

Earn the Cloud Care badge

You can describe an API, model prediction, secret, privacy boundary, latency, inaccuracy, and cost—and pause for adult setup before using a real account.

Explain it back

The AI replies with Python code telling Reachy to move. Should your program run it?

▼ **Explain it first, then compare**

No. Treat the reply as text. Only the local, pre-authored movement policy can choose a permitted gesture.

Want another experiment? Find Mission 7 in the optional challenge bank after the nine missions.

Adult reading

- OpenAI Responses API reference (adult reading)
- OpenAI audio speech API reference (adult reading)

MISSION 8 · 29 MINUTES

Final Mission: Build a Robot Friend

Build one bounded Sense → Understand → Decide → Act turn with two privacy gates and local motion rules.

One thoughtful turn makes a helpful friend!

Today you will make…

A one-turn robot friend and a before/after improvement report.

Bring your earlier skills: sensors, variables, conditions, servos, verification, and privacy.

You are ready to move on when you can:

- Explain the data at all four stages.
- Demonstrate a denied privacy gate without a later upload.
- Propose one improvement and name the test that would show it helped.

Words to discover

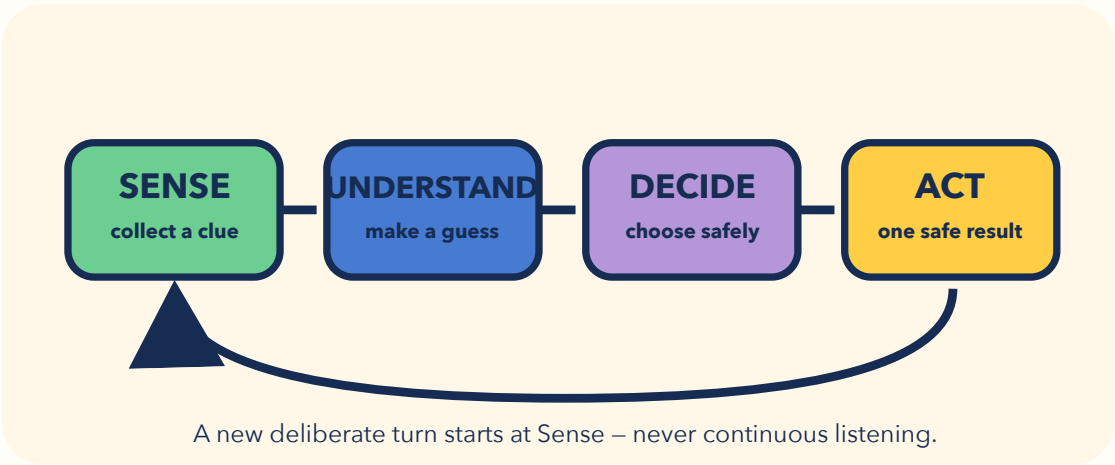
Sense **Understand** **Decide** **Act** **privacy gate** **bounded turn** **verification**

One turn, four stages

Our final friend takes exactly one bounded turn: Sense a deliberate three-second microphone clip, Understand it with transcription, Decide through a short cloud reply plus local keyword rule, then Act by requesting speech and one small gesture. Today's 29-minute plan: 4 minutes to recall the stages, 5 to predict, 12 to build and test one turn, 5 to explain the result, and 3 to collect your badge.

Name the limitation at each stage

Sense means deliberately collecting a three-second sound. Understand means making a fallible transcript. Decide means choosing with a short cloud reply and a local rule. Act means requesting speech and at most one safe, pre-authored gesture. The loop returns to Sense only when a new turn starts; it never listens forever. Sense can capture noise or private audio. Understand can guess the words incorrectly. Decide can receive a slow, wrong, or unsuitable model prediction. Act can fail or be interrupted. These limits are why the program has two explicit privacy gates, bounded time, short output, and adult checkpoints.



A new deliberate turn starts at Sense — never continuous listening.

Read the words beside this diagram too; they are the instructions.

Run one supervised friend turn

With an adult, run this command only after the appendix's cloud safeguards are checked. It connects before the countdown, records one target three-second window, and saves temporary WAVs. Normal exit, errors, and denied gates trigger local cleanup; a crash or power loss can leave files behind. Deleting local files does not erase robot or cloud copies. Two yes/y gates cover three possible cloud requests. The voice is AI-generated, not a human speaking.

Replace practice.py with these four lines and run python3 practice.py. A for loop visits each item in the list once. Keep four spaces before print(stage). It prints the four stages then stops; it does not start the robot. This is your paper plan made into Python.

Tiny Python practice · runs on your Mac

```
stages = ["Sense", "Understand", "Decide", "Act"]
for stage in stages:
    print(stage)
print("One turn finished.")
```

```
python3 examples/mission_08/robot_friend.py
```

Complete runnable file in the companion folder: examples/mission_08/robot_friend.py. The web edition includes the full source and a Copy source button.

Hint: The tiny for loop prints four stages once. It is not continuous listening. Trace the full program using these same stage names.

Trace Sense → Understand → Decide → Act

Before running, name what is sensed, what the transcript might guess, which local gesture keyword could be selected, and how you will verify the sound. Predict what happens when either privacy gate receives `no`.

Keep the movement policy local

Draw four boxes: audio clip → guessed transcript → reply plus gesture name → requested sound and motion. The local rule looks for dance, sleep, or rest in the generated reply; otherwise it chooses WAVE. Thus the reply can influence the gesture name, but cannot supply angles or code. DANCE, SLEEP, and WAVE are tiny pre-authored poses—not full dances or the SDK sleep command. All movement targets stay in Python. Without approved cloud access, trace this turn on paper instead.

Hint: Choose one improvement: clearer words, quieter recording space, or clearer feedback to the user. Keep the prepared motion limits and privacy gates unchanged.

Use the two privacy gates

With your adult, clear the movement area and ask nearby people before recording. At Gate 1, leave Terminal waiting and use Finder → Go → Go to Folder to open the printed temporary folder and listen to its WAV. Decline if unsure. After transcription, review the guessed words at Gate 2. A yes/y there authorizes both cloud requests: transcript to text model, then generated reply to speech model for a WAV. Both can cost money; the reply may repeat the transcript. Tell listeners the voice is AI-generated. A denied gate prevents later cloud, playback, and motion requests; it cannot undo an earlier upload.

Stop without guessing

If recording or cloud setup fails, stop; local temporary audio cleanup is attempted. Each cloud call is bounded with no automatic retries. If motion raises an error or is interrupted, no later movement target is sent. Do not assume the robot is stopped or neutral. Playback and motion may overlap; "play requested" is not proof of sound. If silent, ask an adult to check the audio path.

Adult troubleshooting note: WebRTC must be selected. The local WAV travels over HTTP to the Reachy daemon, and REST requests playback. Check daemon-side uploaded files after a session; local temporary cleanup does not delete those copies.

Hint: Name the first stage whose evidence is missing. Do not rerun every stage automatically: that could upload again, cost more, or repeat a physical action.

Explain one limitation per stage

Write one Sense limitation, one Understand limitation, one Decide limitation, and one Act limitation. Record both privacy answers, the selected local gesture, and how you verified the result without treating the model as a controller.

Earn the Robot Friend badge

You can trace one deliberate turn, explain both privacy gates, and keep generated text separate from executable code and motor angles. Mark each real stage as observed, failed, or not run.

Explain it back

How would you prove your robot friend improved, rather than just looked different?

▼ **Explain it first, then compare**

Choose one observable goal, record the first result, change one thing, and compare using the same test. Report a failure or an unrun test honestly.

Take this skill forward: Next phases can reuse this method: describe the mechanism, make one small change, measure, and explain.

Want another experiment? Find Mission 8 in the optional challenge bank after the nine missions.

Optional challenge bank

Pick one, not all three. Save it for another day if your mission timer is up. Predict, change one thing, test, and explain.

Mission 0 · Motor Detective

easy · Classify a clue: Read one error card and label it network, one motor, several motors, all motors, or daemon ready.

medium · Write a hypothesis: Write: "I think ___ because the real error says ___." Then choose one safe boundary to check.

creative · Create a paper decision tree: Draw a paper decision tree from each clue to one next step. Put "Stop and get an adult" before every physical action.

Mission 1 · Hello, Reachy!

easy · Change the greeting: Print a greeting in your own words.

medium · Use two variables: Store a robot name and your name, then print both.

creative · Robot introduction: Write a three-line introduction that makes Reachy sound welcoming.

Mission 2 · Reachy Moves Carefully

easy · Draw a small move: Draw the neutral head pose and the small head tilt. Label them as whole-head poses, not individual servo angles.

medium · Code the error: In practice.py, write target = 20 on one line, measured = 5 on the next, then print(target - measured). Predict 15 before running. Change measured to 20. This is arithmetic on the Mac, not motor control.

creative · Direct a character: On paper, storyboard a curious and a sleepy greeting using the same small poses but different timing. Explain which comparison would test the character effect. Do not expand the real movement range.

Mission 3 · Reachy Sees Pixels

easy · Find bright and dark: Point to one bright and one dark area in the supplied practice still.

medium · Decode a shape: Explain what height, width, and 3 mean in a BGR shape tuple.

creative · Pixel postcard: Draw a simple pixel picture and label three colors without using a person's face.

Mission 4 · Reachy Speaks

easy · Plan a greeting: Write your own five-word greeting and predict its duration. Changing printed text does not change the supplied WAV.

medium · Compare audio: Explain one way a recorded sound differs from generated speech.

creative · Design a waveform: Draw a waveform for a two-beat robot announcement at a comfortable volume.

Mission 5 · Reachy Listens Carefully

easy · Plan a clip: Write three safe words for a three-second practice clip.

medium · Find a noise source: Name one sound that could confuse recognition and one way to reduce it.

creative · Make a listening sign: Design a sign that says when Reachy is listening, recording, stopped, and still local.

Mission 6 · Reachy Decides with Rules

easy · Pick a branch: Trace what happens when the answer is `hello`.

medium · Add an elif: In your small practice.py program, insert elif word == "maybe": before else. Add one helpful print line indented by four spaces. Test maybe and an unknown word. No robot moves.

creative · Design a kindness rule: Write an if, elif, else plan for three friendly greetings and one safe fallback.

Mission 7 · Reachy Meets Cloud AI with Care

easy · Spot a secret: Explain why an API key belongs with an adult and out of source code.

medium · Improve a safe prompt: Turn a vague made-up robot question into a specific, non-private question under 500 characters.

creative · Create a verify card: Design a card with three steps for checking a model answer and one reminder about cost.

Mission 8 · Final Mission: Build a Robot Friend

easy · Name the stages: Put Sense, Understand, Decide, and Act in order and name one limitation for each.

medium · Trace a reminder: Write one safe spoken input, the guessed transcript, the local gesture, and a possible response. Explain why a cloud reply may differ from your prediction.

creative · Design a friend card: Draw a Robot Friend feature and label its two privacy gates and its local movement boundary.

Quick reference

Your 25-minute mission

2 minutes • Prepare
Open the course folder in Terminal. Run source .venv/bin/activate. Clear Reachy's movement area.

3 minutes • Predict
Read the goal. Say what you expect one piece of code to do before running it.

10 minutes • Try
Run the mission command. Watch Reachy and read Terminal. Compare the result with your prediction.

7 minutes • Experiment
Make one suggested safe change, save with Command+S, and rerun. Keep the original so you can undo your edit.

3 minutes • Remember
Write: I changed __. I expected __. I observed __. Next time I will __. Save remaining challenges for another day.

Terminal or Python?

Terminal command
python3 examples/mission_01/hello_reachy.py tells the Mac to run a saved Python file. Type it in Terminal, then press Return. In the activated course environment, python and python3 use the same toolbox.

Python code
print("Hello, Reachy!") belongs inside a .py file in your code editor. Keep ordinary straight quotation marks.

Folder check
pwd shows your folder. ls lists its files. Look for examples and requirements.txt.

New Terminal window
Run source .venv/bin/activate again. The (.venv) prompt means this window uses the course's package toolbox.

Control+C
Interrupts Python but may not stop a motor command already sent. For unexpected motion, use the physical stop procedure your adult showed you.

Python moves

print("Hello!")
Shows text in Terminal. Quoted text is a string. Printing does not make Reachy's speaker talk.

name = "Reachy"
A variable names a value. = stores a value; == asks whether two values are equal.

import
Brings a toolbox into your program. The Reachy SDK is a toolbox for communicating with the robot.

function()
A named group of steps. Values inside parentheses are arguments: duration=1.5 asks a movement to take 1.5 seconds.

with open_robot() as mini:
Opens an SDK connection called mini. Indented steps use it; leaving closes it. Connecting can configure control settings. with does not mean read-only, and closing is not an emergency stop.

Indentation
Four spaces group steps under a with, if, or def line. Keep the colon at the end of the opening line.

if / elif / else
Choose one branch. If the word is dance, choose DANCE; otherwise choose another known response.

Angles and units
Degrees and radians both measure angles. Keep the lesson's conversions and movement limits; change only the suggested values.

 # comment
A note for the human reader. Python ignores the rest of that line.

Robot clues

motor
Uses energy to move a joint.

sensor
Collects information, such as camera light.

daemon
Background helper that talks to Reachy.

verification
Check that the result matches the plan.

Pixels and samples
A picture is a grid of pixel values. Recorded sound is a sequence of measurements over time.

Rules and AI
A rule follows branches you wrote. A model predicts an answer from learned patterns. Neither is a human brain; a convincing answer can be wrong.

Sense → understand → decide → act
Record a short greeting, guess its words, choose a bounded reply and gesture, then request sound and motion. Check what actually happened; "understand" does not mean human understanding.

Debug loop

Observe
Read the last error line. Write the exact message and the command you ran.

Hypothesize
Make one testable guess: "Maybe I forgot to activate .venv." A guess is not yet a fact.

Test
Try one safe change. Check folder, spelling, quotes, and indentation. For physical faults, stop and get an adult.

Verify
Rerun once and compare. A missing-motor label is a clue, not proof of which wire is loose.

Cloud check
Cloud runs are optional and adult-supervised. Before yes, review the actual clip or text and the under-13 safeguards with your adult. You may always choose no. Never enter a key into this tutorial.

Glossary

actuator
An actuator turns a command into physical action. Each Reachy head motor is an actuator; the six head servos cooperate through arms and rods to position one platform.

algorithm
An algorithm is a set of steps for solving a problem. Reachy can follow an algorithm for choosing a friendly greeting.

API
An API is a careful doorway that lets one program ask another program for help. An adult might use an API to connect Reachy to an approved cloud helper.

Boolean
A Boolean is a value that is either true or false. Reachy can use a Boolean to remember whether a practice button was pressed.

bus
A bus is a shared communication path. Reachy's motor messages use electronic IDs so the intended servos can respond.

camera frame
A camera frame is one picture collected by a camera at one moment. Reachy can inspect a local camera frame without uploading it.

cloud
The cloud means computers you use through the internet instead of the computer beside you. Reachy needs adult approval before sending anything to a cloud service.

condition
A condition is a question a program can answer true or false. Reachy can ask whether a practice answer is yes before choosing a greeting.

coordinate
A coordinate describes position relative to a frame. For Reachy's head, +x is forward, +y is left, and +z is up. A coordinate is not automatically a safe target.

daemon
A daemon is a background program. Reachy's daemon communicates with hardware and serves requests from other programs; the status reader asks it for information.

debug
To debug is to find and fix a problem using clues and tests. A Reachy detective debugs by checking one safe clue at a time.

feedback
Feedback means using a measurement to guide the next correction. A Reachy servo measures its shaft position; it does not simply assume it arrived.

function
A function is a named bundle of program steps that can be used again. A Reachy greeting function can print the same friendly welcome whenever it is needed.

input
Input is information a program receives. Reachy can receive a safe practice button press or a three-second audio clip as input.

linkage
A linkage carries mechanical force between moving parts. Reachy's servo arms and rods connect the head servos to a shared platform; they are not electrical wires.

loop
A loop repeats a set of steps. Reachy could use a loop to check a practice sensor several times, with adult supervision.

model
A model finds patterns to make a likely guess or response. A cloud model might suggest words for Reachy, but a person still checks them.

motor
A motor turns electrical energy into mechanical motion. Use Reachy's prepared small-motion lessons with clear space after adult setup; faults and repairs need an adult.

motor ID
A motor ID is an electronic address. In the pinned Reachy configuration, stewart_1 has ID 11. A sticker number and an ID need not be the same.

output
Output is what a program sends out, such as text, sound, or light. Reachy can give a kind audio greeting as output.

pixel
A pixel is a tiny colored square that helps make a digital image. Reachy's camera frame contains many pixels, not instant understanding.

pose
A pose describes position and orientation. Reachy's head pose is not the same as an individual motor angle.

program
A program is a set of instructions a computer follows. A Reachy program can print a greeting or follow a simple rule.

sensor
A sensor collects information from the world. Reachy's camera is a sensor that collects light in a frame.

servo
A position servo combines a motor, gears, a position sensor, and a controller. Reachy's servos compare their measured shaft position with a target.

string
A string is text inside a program, such as words or a sentence. Reachy can store its friendly greeting as a string.

target
A target is a desired result, not proof of success. Reachy receives a pose target; the learner still observes what happened.

torque
Torque is turning force. Reachy's servo gears trade speed for torque; resisting a servo by hand is not a safe experiment.

variable
A variable is a labeled place where a program remembers a value. Reachy can keep a chosen robot name in a variable.

verification
Verification means checking that a result is what you expected. After a small Reachy movement, you verify the pose before changing anything else.

Adult appendix

The independence handover

After adult setup, learners aged 10–11 can follow the local missions independently, including the prepared small movements. Optional cloud transcription and Missions 7–8 require an adult present. Allow 20–30 minutes per mission; save an optional challenge for another day.

Demonstrate starting Reachy Mini Control, running the health check, clearing the movement area, and using the physical power switch. Agree where the learner can find you when a STOP message appears. Keep hands, hair, drinks, and loose objects outside the movement area.

Cable work, opening the robot, changing motor IDs, firmware updates, and repairs remain adult tasks. Ctrl+C interrupts Python but is not a guaranteed emergency motor stop.

Software checks do not certify your robot as healthy. Check each physical capability during setup before independent use. This course does not claim that every example has been tested on your hardware.

Mac setup: a private Python toolbox

Install Python 3.12 using the official Python installer or Reachy installation guide. Open Terminal with Command+Space, type Terminal, and press Return. Type cd followed by a space, drag the course folder from Finder into Terminal, and press Return. The correct folder contains requirements.txt and examples.

Run each command separately. Creating the environment and installing packages are one-time adult steps. A virtual environment keeps this course's packages together.

```
python3.12 --version
```

```
python3.12 -m venv .venv
```

```
source .venv/bin/activate
```

```
python -m pip install -r requirements.txt
```

```
python -c "import sys; from importlib.metadata import version; print(sys.version.split()[0]); print(version('reachy-mini'))"
```

Expect Python 3.12.x and SDK 1.10.0. This course pins reachy-mini==1.10.0. Confirm the robot daemon version in Reachy Mini Control or its status display. Resolve any version mismatch before teaching, then recheck motion and media. Do not ask the learner to upgrade packages at random.

Official installation: https://huggingface.co/docs/reachy_mini/SDK/installation

Network and media readiness

The examples connect to reachy-mini.local over the local network. Put the Mac and Wireless robot on the same reachable network. Guest-network isolation, VPN routing, or a blocked local-network permission can prevent discovery. Internet access and access to the robot are separate checks.

python examples/mission_00/check_robot.py

python examples/mission_01/hello_reachy.py

Mission 0 uses a read-only HTTP status request. Mission 1 reads antenna positions without a movement target, but the SDK 1.10.0 constructor sends its automatic-body-yaw setting to the daemon. Do not describe that SDK connection as strictly read-only. Keep other movement apps inactive. Proceed only when status and physical checks look normal; a startup error does not identify a loose cable.

Camera and microphone examples select WebRTC. The official GStreamer guide provides macOS dependencies as Python wheels; Linux apt commands do not belong on the Mac. If motion works but media fails, check the media setup and daemon stream separately.

During setup, try a toy photograph, a quiet speaker clip, and a three-second non-private recording. Open the saved image and listen to the sound. A printed “request sent” is not proof of playback.

Media architecture: https://huggingface.co/docs/reachy_mini/SDK/media-architecture

macOS media dependencies: https://huggingface.co/docs/reachy_mini/SDK/gstreamer-installation

Cloud accounts and costs

Missions 0–4 and 6 need no AI account. Mission 5 can record locally and skip transcription. For cloud demonstrations, review current eligibility, parent/guardian consent, data rules, and spending first. The child does not create an account or handle credentials. Remain present for uploads and model replies; a friendly prompt and length cap are not a content-safety filter.

Under-13 checkpoint: OpenAI guidance says not to process personal data of children under 13 (or the applicable age of digital consent) without first implementing Zero Data Retention. A child's voice can be personal data even without names. An adult-owned key, invented phrase, typed yes, or store=False does not implement this. Verify eligibility and approved data controls for the actual project and endpoints before using a child's data; otherwise keep recordings local and use the paper or simulated activities.

Use an adult-owned API project. Privately set OPENAI_API_KEY in the Terminal session that launches Python, or use an adult-managed launcher or secret manager. README includes a hidden-input zsh command. Never paste a key into Python, HTML, browser storage, screenshots, or learner logs. These examples do not automatically load a .env file.

Check available models and current pricing before a cloud session. Defaults: gpt-5-mini for text, gpt-4o-mini-transcribe for transcription, and gpt-4o-mini-tts for speech. An adult can configure tested compatible alternatives with OPENAI_TEXT_MODEL, OPENAI_TRANSCRIBE_MODEL, and OPENAI_SPEECH_MODEL.

Agree on a small number of attempts and monitor actual usage. Budget alerts may not immediately stop spending. A text question makes one request; a full Robot Friend turn can make three: transcription, text, and speech.

The client uses a 20-second request timeout and no automatic retries. It is not a strict whole-program deadline. Text generation has a 1,024-token cap including reasoning, with minimal reasoning for the GPT-5 Mini default; the displayed answer is separately limited to two sentences and 280 characters. This gives more room than the old 80-token cap, but can still produce an incomplete response. Incomplete or empty replies fail without downstream speech or motion. Check usage: hidden reasoning is billable too.

A timeout does not prove a request was never processed or billed. HTTP 429 can indicate request-rate or account-quota limits. Avoid repeated retries. The examples are small supervised teaching demos, not a fully moderated child-facing product; reassess model safety, filtering, and monitoring before wider use.

Account guidance: https://developers.openai.com/api/docs/guides/production-best-practices

Current pricing: https://openai.com/api/pricing/

Under-18 guidance: https://developers.openai.com/api/docs/guides/safety-checks/under-18-api-guidance

Reasoning and incomplete responses: https://developers.openai.com/api/docs/guides/reasoning

What leaves the Mac?

The camera lesson saves a local photograph without uploading it. The listening lesson saves a WAV and uploads it only after the typed transcription confirmation. Practice with toys and invented greetings; exclude names, schools, addresses, faces, passwords, and other people's conversations.

Mission 7 sends the typed question. Mission 8 asks to upload its clip for transcription, then asks to send the recognized words to a text model and the generated reply to a speech model. Teach the learner to decline when unsure. Adult setup does not replace checking the actual content.

Mission 8 attempts to remove temporary Mac WAVs after the turn. A crash or forced termination may leave files. Playback uploads a WAV to the robot: Mac cleanup does not delete robot-side or cloud copies. Inspect daemon media storage after teaching. Mission 3 photographs and Mission 5 recordings remain on the Mac until removed, and another run overwrites their fixed filenames. Rename a copy before a comparison; exclude all learner media from distribution.

Clearly tell listeners that Mission 8's voice is AI-generated, not a human. Its local substring rule inspects the generated reply to choose WAVE, DANCE, or SLEEP. This is a bounded text-to-gesture policy, not semantic understanding: "forest" even contains "rest." Keep the tiny fixed targets unchanged; do not treat the gesture name as evidence of correct intent.

The text request uses store=False, which does not remove all provider retention or abuse-monitoring storage. Review the provider's data controls. AI can be wrong, and a short-answer prompt cannot guarantee suitable content. Teach the learner to stop and tell an adult about an upsetting or confusing reply.

Data controls: https://developers.openai.com/api/docs/guides/your-data

Troubleshooting without guessing

"No such file" usually means the wrong folder: run pwd and ls and look for examples and requirements.txt. "ModuleNotFoundError" often means .venv is inactive or installation is incomplete. Activate the environment first; an adult can rerun the requirements installation if needed.

If reachy-mini.local is not found, check the network and robot address in Reachy Mini Control. If it is reachable but the daemon reports an error, preserve that exact status. These are different problems.

A missing motor can indicate a connector, power, cable, or motor problem. Its label alone does not identify the faulty connection. Stop commands and power off before an adult inspects connectors using the manufacturer's procedure.

For heat, binding, unusual noise, or unexpected motion, use the demonstrated physical stop procedure and get an adult. Save the mission, command, and exact visible clue without secrets or private recordings.

Official troubleshooting: https://huggingface.co/docs/reachy_mini/troubleshooting

Servo and 3D learning model: what is verified

Use the HTML Reachy lab for a two-minute mechanism exploration or optional Python rehearsal. The viewer uses SDK 1.10.0 URDF/STL geometry plus clearly schematic electronics and servo internals. It never connects to a robot. Start it with npm run serve from the full project: the Python worker requires that server's restrictive security header. A copied dist folder is the public output, not the Node project; an adult must keep the supplied server running. No CDN or account is required.

The Wireless assembly guide connects head motors 1–2, 2–3, 4–5, 5–6, and then 3–4 (steps 13–16 and 21, checked 9 September 2026). A two-branch drawing that omits 3–4 is not an accurate harness guide. The teaching viewer shows these neighbor links but does not reconstruct controller, body, or antenna entry paths. Its optional all-nine hub diagram is logical communication, not physical star wiring.

Wire colors are semantic colors, not connector pinouts. Exploded offsets are not a teardown sequence. Explicit CAD-instance mapping links casings to the URDF joints and default software IDs; it is not a scan of your unit. The drawing cannot establish collision clearance, safe physical angles, actual board dimensions, or servo gear layout. Repairs require the correct manufacturer guide.

Browser Python uses the small reachy_sim adapter, not the installed robot SDK. Its create_head_pose defaults to degrees and metres like SDK 1.10.0; degrees=False uses radians and mm=True uses millimetres. Body and antenna inputs remain radians. The adapter returns a teaching dictionary rather than the SDK's 4×4 matrix. get_pose returns planned targets, not sensor measurements; look and listen use pretend inputs, and say only displays text. Downloaded lab code does not run as a hardware program.

The servo lab uses a deliberately simple half-error rule to explain feedback. It is not the actual servo controller. The motion cartoon illustrates timing, not exact kinematics. Ask learners to distinguish what the model shows, what they measured, and what they have not tested.

Phase 1 assessment: have the learner explain feedback; distinguish mechanical linkages from electrical cables; change one Python variable; report one failed or unrun test honestly; and explain why cloud text cannot become motor commands. Future MicroDuck and robot-arm phases can reuse these habits without copying Reachy-specific joint angles or wiring.

Assembly source: https://huggingface.co/spaces/pollen-robotics/Reachy_Mini_Assembly_Guide

Geometry source: https://github.com/pollen-robotics/reachy_mini/tree/v1.10.0/src/reachy_mini/descriptions/reachy_mini