PADMA ROBOTICS LAB — TERM 1 LESSON PLANS

Term 1 · Signals · Weeks 1–10 · Grades 6–8

Format: 10 × 40-minute lessons · **Devices:** 1 per 4 students (Windows or Android tablet) **Hardware:** the robot is present from Week 1 but **programmed only in Week 8**

How to use this pack

Every lesson is the same shape. Students learn the rhythm by Week 2, and a substitute teacher can deliver any lesson from this document alone.

SEGMENT	TIME	CUMULATIVE
Warm-up (unplugged)	5 min	5
Guided Instruction (teacher demo)	10 min	15
Independent Practice (studio)	20 min	35
Debugging & Discussion	5 min	40

Two rules that protect the schedule

1. **Do not open devices before minute 15.** The unplugged segment is where the concept is actually formed. Teachers who skip it always run out of time later.
2. **If you are running long, cut the guided demo, never the debugging.**

Blocks used this term

Padma is Scratch-based, so these are standard Scratch 3 blocks. Confirm the exact wording in your build once, then tape this table to the wall.

THIS PACK SAYS	WHAT IT DOES
`when ⚑ clicked`	Starts the script
`when [key] pressed`	Keyboard trigger
`when this sprite clicked`	Tap trigger
`broadcast [msg]` / `when I receive [msg]`	Message between sprites

THIS PACK SAYS	WHAT IT DOES
`move [10] steps`	Forward
`turn [15] degrees`	Rotate
`go to x: y:`	Jump to a point
`repeat [10]`	Counted loop
`wait [1] seconds`	Pause
`set [power] to [100]`	Assign a variable
`change [power] by [-10]`	Update a variable
`show variable [power]`	Display it on the stage
`pen down`	Draw as you move
`switch costume to [__]`	Change appearance
`stop other scripts in sprite`	Halt without stopping everything

Classroom roles (1 device : 4 students)

Rotate weekly so nobody is permanently the typist.

ROLE	RESPONSIBILITY
Pilot	Holds the device, drags the blocks
Navigator	Reads the challenge card, checks the plan
Recorder	Writes the notebook page and the bug log
Reporter	Presents at the end; answers the teacher's questions

Standard materials (every week unless noted)

Devices charged ≥80% · Padma installed, network off · printed challenge card · debug card (laminated, one per table) · notebooks · exit tickets · **the robot in its box, powered off, on the teacher's desk** · one spare device.

File naming: `P1_01_FirstScript`, `P1_02_Shapes`, … `P1_10_Console`. Enforce it in Week 1 — you will need it in Week 9.

WEEK 1

Meet Padma, Meet the Robot

1. Lesson Info Block

Topic	What a program is; the four zones of Padma; the first script
Duration	40 minutes
Keywords	program · instruction · algorithm · event · sprite · stage · trigger
Mastery statement	*"I can make something on screen respond to what I do."*
Materials	Devices (1:4) · Padma open to a blank project · 8 challenge cards · **the robot, boxed, on your desk** · 1 marker (for the warm-up) · exit tickets · notebooks

2. Teacher Preparation Checklist

- [ ] Launch Padma on the **oldest device you own**, in airplane mode. Drag one block, run, close, reopen. If it fails here it fails in class.
- [ ] Confirm Padma opens a blank project by default, or pre-load `P1_00_Blank`.
- [ ] Display brightness to maximum; disable auto-lock for 45 minutes.
- [ ] Put the robot's box on your desk, **lid visible, label facing the class**. Do not open it. Do not explain it yet.
- [ ] Write on the board: **"A computer does exactly what you say. Not what you mean."**
- [ ] Arrange tables in 4 clusters; one debug card each; role sticky notes prepared.

3. Step-by-Step Instructional Guide

WARM-UP — "I AM THE ROBOT" (5 MIN)

Hold a marker, stand at the front.

> **Script:** "For five minutes I am not a teacher. I am a machine. I only do exactly what you tell me. Nothing more. Who can get this marker into my hand?"

Take instructions literally and wrongly. "*Pick up the marker*" → mime grabbing it with your elbow. "*Move your hand to the marker*" → move to the wrong side.

After two or three attempts someone will say "*move your right hand forward ten centimetres, close your fingers, lift.*" Do it.

> **Script:** "Why did that work? Because it was **precise**. The machine wasn't being difficult. It has no idea what you *mean*. It only knows what you *said*. Every day this term, that is the bug you will be fixing."

Write on the board: **precise** → **predictable**.

Then tap the robot box once.

> **Script:** "That box stays shut for seven weeks. By the time we open it, you'll be able to make it do what you tell it. Not what you mean. What you tell it."

GUIDED INSTRUCTION — FOUR ZONES, ONE HAT (10 MIN)

Step 1 — name the zones (2 min). Point and say, students repeat: **stage** (where things happen), **sprites** (the cast), **palette** (the toolbox, grouped by colour), **scripts** (the workbench).

Step 2 — the hat rule (3 min). Drag `when this sprite clicked` into the script area. It has a curved top.

> **Script:** "The curved top means it's a **hat**. A hat is a **trigger** — it waits for something to happen. Blocks only run when a hat tells them to. No hat, no program. This is the most common reason nothing happens."

Attach `say [Hello, Padma] for [2] seconds`. Run. Tap.

Step 3 — break it on purpose (3 min). Drag a bare `say [I am broken]` onto empty space. Tap the sprite. Nothing.

> **Script:** "Why nothing? … No hat. No trigger. Write it down: **hat = trigger = nothing runs without one.**"

Step 4 — second event (2 min). Add `when [space] key pressed → say [Signal received]`.

> **Script:** "One sprite. Two triggers. Different response to each. That is already a real program."

INDEPENDENT PRACTICE — STUDIO TASK: GREETING PROTOCOL (20 MIN)

Task A (6 min). Make the sprite say its own name and mission for 2 seconds when tapped.

Task B (6 min). Press **space** → say `"Signal received."`

Task C (8 min). Add a **second sprite**. Each has its own tap event and says something different. Tapping one must not wake the other.

Save as `P1_01_FirstScript`.

Circulate constantly. At minute 10, every team must have at least one working hat block. Any team without one gets your full attention for 60 seconds — everything downstream depends on it.

Extension on the card back: three sprites, keys 1/2/3, each reports in sequence.

DEBUGGING & DISCUSSION — "WHERE IS THE TRIGGER?" (5 MIN)

Project a student's non-working project (ask permission) or your prepared broken example.

Run two questions:

1. **"What did you expect?"** → "It should talk when I tap."
2. **"Where is the trigger?"** → silence, then hands.

> **Script:** "Three failures you will meet this term. One: **no hat** — nothing runs. Two: **wrong sprite** — you wrote it on the satellite but you're tapping the rover. Three: **the app isn't in run mode** — you're still in edit mode."

Exit ticket: *"A hat block is…"*

4. Differentiations

Supporting struggling students

- Pre-place the hat block in `P1_01_Starter` so their first success comes within 60 seconds.
- Provide a **phrase bank** of four pre-written lines on the card.
- Make them **Navigator** rather than Pilot in Week 1 — reading the card aloud is a real, assessable contribution.
- Accept a spoken explanation in place of a written notebook entry.
- If a student freezes, give **one instruction, not a question**: "Drag the gold hat block to the left edge."

Advanced extension challenges

- **Crew roll call:** three sprites, three keys, sequential reports — the *sequence* must be enforced by the student, not the app.
- **Mission clock:** a countdown using `wait` between lines.
- **Costume response:** the sprite changes appearance on tap *as well as* speaking — this previews state in Week 5.
- **Write the card:** design a challenge card for an absent partner that works without verbal help.

WEEK 2

Precision Drawing: Sequence and Parameters

1. Lesson Info Block

Topic	Sequence, order, parameters, angles — **and the first flowchart**
Duration	40 minutes
Keywords	sequence · order · parameter · step · degree · turn · pen · closed shape · **flowchart** · terminator · decision
Mastery statement	*"The shape is decided by the numbers — and I can draw it before I build it."*
Materials	Devices · printed studio task · **8 instruction card sets** (6 cards each, cut, one per table; two sets deliberately wrong) · angle reference strips · **flowchart symbol poster (A3, wall)** · exit tickets

2. Teacher Preparation Checklist

- [ ] Print and cut 8 six-card sets. Two must be wrong (`turn 45` instead of `turn 90`) — mark the back with a pencil dot so you know.
- [ ] **Print the flowchart symbol poster** — oval, rectangle, diamond, parallelogram, arrow — and tape it to the wall. It stays there for three terms.
- [ ] **Test** `pen down` **in Padma on your demo device.** Some builds require enabling the pen category first. Do this before class.
- [ ] Prepare `P1_DEMO_2` with a correct square and a square with a gap at the last corner.
- [ ] Draw a large square on the board with one corner open and "90°" at each turn.
- [ ] Print angle reference strips: 90° / 60° / 120° / 45°.

3. Step-by-Step Instructional Guide

WARM-UP — "INSTRUCTION SHUFFLE" (5 MIN)

Hand each team a sealed card set, face down. Do not let them look until "go."

> **Script:** "Six cards describing how to draw a shape. Ninety seconds to order them — on the desk, not the device. Then one person follows your order exactly, on the board, and we see what appears."

Two teams will produce a shape that doesn't close. Do not correct them.

> **Script:** "Team 4's shape has a gap. Team 6's doesn't. Same cards, same order. The only difference is one number on one card. **The shape is decided by the numbers.**"

GUIDED INSTRUCTION — BUILD A SQUARE, THEN BREAK IT (10 MIN)

Step 1 (3 min). Build live: `pen down` → `move [100]` → `turn [90]` → repeat three times manually.

> **Script:** "Four sides, four turns. Watch the turn — it does **not** move the sprite. It only changes which way it faces. Say that back to me."

Step 2 (2 min). Change `100` to `40`. Run.

> **Script:** "Smaller. Still a square. The **size** changed, the **shape** didn't. That number is a **parameter** — the dial on the block."

Step 3 (2 min). Change every `90` to `80`. Run. The square fails to close.

> **Script:** "Four turns of 80 is 320. A full circle is 360. It's 40 degrees short. Nothing went wrong — the machine did exactly what we said, and what we said was 40 degrees short."

Step 4 — the flowchart (3 min). Draw on the board, using the four shapes:

SHAPE	MEANING
Oval	Start / stop
Rectangle	A step — one block
Diamond	A decision — `if` / `else`
Parallelogram	Input or output — `ask` / `say`
Arrow	What happens next

Draw the square as a flowchart: **oval "start"** → rectangle `pen down` → rectangle `move 100` → rectangle `turn 90` → four boxes back to the move → oval "stop".

> **Script:** "This is the same program, drawn without a computer. Nobody can argue about which app you used. **A flowchart is a program that works everywhere.** In eleven weeks you'll write this in Python, and this picture will still be right."

> **Script:** "Notice what's missing. There is no diamond. No decision. This program can only go one way — that's what a **sequence** looks like as a picture."

Point at the four shaded shapes on the wall (print them; they stay up all year).

INDEPENDENT PRACTICE — STUDIO TASK: SHAPE LADDER (20 MIN)

Students climb five rungs. **No skipping**, but fast teams move quickly.

RUNG	TASK	TIME
0	**Draw the flowchart** of your square, before you build anything	3 min
1	Square, 100-step sides — build it **from your flowchart**	3 min
2	Square, 50-step sides, different colour	2 min
3	**Triangle** with equal sides — flowchart it first	6 min
4	A letter of your name, straight lines only	4 min
5	Hexagon	2 min

Rung 0 is not optional and it is not decoration. A student who can draw the flowchart can rebuild the program on any machine, in any language, in five years.

Rung 3 is the discovery moment. Most will try 60° and get an open zigzag. Let them. When a team closes it, ask them to explain *why* 120 works and 60 doesn't.

Circulate with questions only: "How many sides?" → "How many turns?" → "How many degrees in a full circle?" → "So how much per turn?"

Save as `P1_02_Shapes`.

DEBUGGING & DISCUSSION — "THE GAP" (5 MIN)

Show your open square.

1. **Predict:** "Will it close?"
2. **Observe:** run it.
3. **Isolate:** "Which block is wrong — move or turn?" Change one turn to 90; two sides get longer. Narrow it to the turns.

> **Script:** "This is **isolating the fault** — change one thing, watch what changes. In Week 9 you'll do it properly. For now: don't change five numbers hoping one is the answer."

Second failure: a stray `pen up` leaves a line across the shape, or a missing `pen down` hides the first side.

Exit ticket: *"A triangle needs three turns of ___ degrees because…"*

4. Differentiations

Supporting struggling students

- Give the angle strip and a printed trace page showing a square with arrows.
- Reduce the ladder to rungs 1 and 4.
- Sit with them for the first 90 seconds of rung 1 and have them read the blocks aloud.
- Permitting copying for rung 1 is fine; require rung 2 from memory.

Advanced extension challenges

- **Star:** five points using `turn 144` — work it out from "five points, two full turns."
- **House:** square plus triangle roof without lifting the pen between them.
- **Angle detective:** given an irregular shape drawn on paper, compute the turns from the picture.
- **Same letter, two sizes** using the same block sequence, and explain in writing which number changed.

WEEK 3

The Stage as a Map: Coordinates and Heading

1. Lesson Info Block

Topic	Position on a plane: x/y, heading, route planning
Duration	40 minutes
Keywords	coordinate · axis · origin · heading · waypoint · route · tolerance
Mastery statement	*"I can tell something exactly where to go."*
Materials	Devices · printed studio task · **masking tape 6×6 floor grid** (set up the day before) · route cards (3-waypoint and 5-waypoint) · exit tickets

2. Teacher Preparation Checklist

- [ ] **Tape the 6×6 floor grid the day before.** Use a different tape colour for the centre lines (x = 0, y = 0). Label the axes with paper arrows. Twenty minutes of work that pays for itself.
- [ ] Pre-load `P1_03_GridTemplate` with a grid backdrop so coordinates are visible.
- [ ] Prepare `P1_DEMO_3` with `go to` and `glide` side by side.

- [ ] Print route cards: **A** (3 waypoints), **B** (5 waypoints, no obstacles), **C** (5 waypoints plus two blocked squares).
- [ ] Write the four quadrant signs on the board: (+,+) · (−,+) · (−,−) · (+,−).

3. Step-by-Step Instructional Guide

WARM-UP — "HUMAN ROVER" (5 MIN)

Call one student to the floor grid, standing at (0,0) facing up.

> **Script:** "This is our robot, and it cannot see. It only obeys. The target is at (4,2). You are the navigation team. Every instruction must be either *move forward N squares* or *turn N degrees*."

Let them overshoot.

> **Script:** "Mission failed. Not because the robot is stupid — because the route was wrong. Notice something: '*go to (4,2)*' is much easier to say than '*forward 4, turn 90, forward 2.*' Today you get both. The second is more powerful; the first is easier to plan."

GUIDED INSTRUCTION — TWO WAYS TO MOVE (10 MIN)

Step 1 — coordinates (4 min). Point at (0,0).

> **Script:** "This is the **origin**. Its name is (0,0). Every other point has a name — two numbers. The first is how far **right**. The second is how far **up**. Left of centre is **minus**. Below centre is **minus**."

Place the sprite at (0,0); then `go to x: 3 y: 2`, then `-3 y: 2`, then `-3 y: -2`.

Step 2 — heading (3 min). Use `point in direction [90]`. **State your build's convention out loud** — this differs between builds and is a genuine source of confusion.

> **Script:** "Direction is not position. Position is *where*. Direction is *which way you face*. You can be in one place facing four ways."

Step 3 — `go to` vs `glide` (3 min). Run both.

> **Script:** "`go to` teleports. `glide` travels. For a *simulation*, either is fine. For a *robot*, there is no teleporting — the machine has to physically get there. Remember that next term."

INDEPENDENT PRACTICE — STUDIO TASK: ROUTE TO THE TARGET · CARD 2 (20 MIN)

Rules on the card:

- Start at `(0,0)`
- Visit every waypoint in order
- May not pass through a blocked square
- End within **one square** of the target (name this: *tolerance*)
- Must run from **one tap** — no manual steering

Route B (core): `(0,0) → (2,3) → (5,3) → (5,-1) → (-1,-2)`, target at `(2,0)`; blocked at `(3,1)` and `(4,-2)`.

Students should:

1. **Draw the route on paper first** — the card has a blank mini-grid. Five minutes that save fifteen.
2. Mark blocked squares their straight line crosses.
3. Add waypoints to go around.
4. Build and run.

> **Script:** "This is **decomposition**. The task is not *reach the target* — that is one impossible instruction. The task is four small journeys, in order, and each one is easy. You are not writing one big program. You are writing four short ones and joining them."

Save as `P1_03_Route`.

The same failure repeats: a straight line between two waypoints crosses a blocked square. The fix is not code — it is **an extra waypoint**. Ask: "Is there a point you could stop at that would prevent this?"

DEBUGGING & DISCUSSION — "WHERE DID IT GO?" (5 MIN)

Project a route that ends in the wrong place. Find it on the board grid.

Three errors to name:

1. **Sign error** — `x: -3` typed as `x: 3`. "One character."
2. **Y-axis confusion** — screen coordinates often go down. State your build's convention and post it.
3. **Sprite centre** — the position is the *centre*, not the corner. An off-centre costume lies about where it is.

Exit ticket: *"At (2,3), want (-1,3). Which number changes, and to what?"*

4. Differentiations

Supporting struggling students

- Use **Route A** (3 waypoints, no obstacles).
- Provide a printed grid with every square's coordinates already written in.
- Teach the one-axis-at-a-time method: fix x in one `go to`, then y in another.
- Pair with a strong Navigator who traces the route with a finger.

Advanced extension challenges

- **Efficient route:** fewest `go to` blocks for all five waypoints — then argue in writing why fewer blocks isn't always better.
- **Return to base** after a key press.
- **Two sprites**, different routes, both avoiding obstacles.
- **Smooth motion** with `glide`, then explain the trade-off in one sentence.
- **Write a route card** that is solvable but not obvious.

WEEK 4

Mission Control: Events and Messages

1. Lesson Info Block

Topic	Multiple events on one sprite; broadcasting between sprites; **first contact with KanKaung**
Duration	40 minutes
Keywords	event · trigger · key press · broadcast · receiver · message name · control panel · remote · hardware
Mastery statement	*"Two sprites can talk to each other — and one of them can move a robot."*
Materials	Devices · printed studio task · paper "remote" (4 labelled buttons per team) · `P1_04_ButtonKit` pre-loaded · **4 KanKaung robots, paired and charged** (half the tables) · masking tape for the driving area · exit tickets

> **Robot week (first of two).** Half the tables drive KanKaung today; the other half in Week 6. Every student drives it once this term. See the rotation note below.

2. Teacher Preparation Checklist

- [ ] Build `P1_DEMO_4` **before class**: a rover sprite plus a status-text sprite wired with one working broadcast.
- [ ] Confirm the broadcast block wording in Padma (`broadcast [__]` + `when I receive [__]`).
- [ ] Pre-load `P1_04_ButtonKit` — four button sprites, named `UP`, `DOWN`, `LEFT`, `RIGHT`, positioned in a cross. **This removes 10 minutes of drawing.**
- [ ] Decide and write on the board which key moves which way.
- [ ] Print 8 paper remotes for the warm-up.

- [ ] Prepare one deliberately broken broadcast (message name spelled differently in sender and receiver).

Robot rotation — read before the lesson

- [ ] **Pair all four robots to their devices before students arrive.** Never pair in front of a class.
- [ ] Tape a **driving area** at the front or side: 1.5 m × 1 m, bounded by books or tape. The robot drives inside it and nowhere else.
- [ ] Write the rotation on the board: **Tables 1–4 drive today; tables 5–8 drive in Week 6.**
- [ ] Cap the robot's speed for this lesson at **40**. Remote control at full speed ends in stripped servo gears.
- [ ] Confirm the **watchdog** is active — the robot must stop within half a second of losing its command stream. Test it by switching off the device while the robot drives.
- [ ] Have one spare robot in the room. If pairing fails, swap and move on; do not debug for more than 60 seconds.

3. Step-by-Step Instructional Guide

WARM-UP — "THE PAPER REMOTE" (5 MIN)

Give each team a paper remote; appoint one student as "the robot."

> **Script:** "I press ▲, the robot stands. ▼ sits. ◀ faces left. ▶ faces right. Ready?"

Press random buttons for 30 seconds.

> **Script:** "What did the remote do? … It didn't move the person. It sent a **signal**, and the person responded. Every button is an **event**: something happens, something else reacts. That's what you're building today."

GUIDED INSTRUCTION — FOUR KEYS, THEN ONE MESSAGE (10 MIN)

Step 1 (4 min). Build `when [UP] clicked → move [20] steps`. Add three more.

> **Script:** "How many hats does this sprite have? Four. A sprite can have as many triggers as you want. Each is a separate little program sharing one character."

Step 2 (2 min). Drive it off the stage edge.

> **Script:** "The buttons work. The sprite is gone. Buttons only send *signals* — they don't know anything about the world. That's next week's problem."

Step 3 — broadcast (4 min). On the rover: `when I receive [arrived] → say [Checkpoint reached] for [2]`. On the status sprite: `when [space] key pressed → broadcast [arrived]`.

> **Script:** "Did the rover receive a key press? No. It received a **message**. A message goes to *everyone*. This is how a real control room works — one signal, many systems responding."

Emphasise **exact spelling**: the name in `broadcast` and `when I receive` must match character for character.

INDEPENDENT PRACTICE — STUDIO TASK: BUILD MISSION CONTROL (20 MIN)

TASK	REQUIREMENT	TIME
A	Four buttons move the sprite up/down/left/right by 20 steps	6 min
B	A fifth key broadcasts `status`; a **separate** sprite updates its text	6 min
C	**Robot tables:** swap your sprite for KanKaung — four buttons drive the real machine	8 min
C′	**Non-robot tables:** pressing `E` broadcasts `emergency`; two sprites respond	8 min

Save as `P1_04_ControlPanel`.

The conceptual leap in B: the status sprite must know something the rover knows. Students will try to put everything on one sprite. Push toward separate sprites — that is where messaging earns its place.

Running the robot table (8 minutes, strictly timed)

1. **(1 min)** Navigator reads the four button names aloud. Pilot takes the device. Recorder holds the stopwatch.
2. **(5 min)** Drive. Direction, stop, direction. Nothing else.
3. **(2 min)** The **test**: hand the device to another team's Navigator and say nothing. If they can drive it without asking, the interface works.

> **Script (before they start):** "This is the same program you just built. Four hats, four messages, one receiver. The only thing that changed is where the commands go. Your buttons now move four kilos of robot instead of a sprite."

> **Script (after):** "Put your hand up if it went somewhere you didn't expect. … That's not a broken robot. That's a robot that has been listening to you exactly. It has no idea where the walls are."

The line to leave with them: *next term you'll write the rule that stops it going into the wall.*

Rotation note. Tables 1–4 drive today and do task C′ as homework or in Week 5's warm-up. Tables 5–8 do the reverse. Every team drives once before Week 8.

DEBUGGING & DISCUSSION — "NOBODY HEARD THE MESSAGE" (5 MIN)

Show `broadcast [Arrived]` but `when I receive [arrived]`.

> **Script:** "Nothing happens. Nothing is *broken*. So where do we look? Not at the blocks that work — at the **join** between them. Capital A, capital A. One character apart and the whole system is silent."

Second failure: **two hats on the same key**. "Ask: how many things listened?"

Third, only if a robot table ran: **the robot kept going after the tablet slept.** Point at the watchdog.

> **Script:** "It stopped. Why? Because we told the firmware: if you stop hearing commands, stop the motors. That is a machine that knows how to fail safely. Remember that — you'll build it yourself in Term 3, and it will be worth more marks than speed."

Exit ticket: "*Why is a broadcast better than a button press? One sentence.*"

4. Differentiations

Supporting struggling students

- Two buttons only (left/right); status sprite changes costume instead of text.
- `P1_04_ControlPanel_Starter` with all four button sprites placed and one script complete.
- Write the exact message names on a sticky note on their device — spelling is the dominant failure here and it isn't a computational-thinking failure.
- Let them dictate the direction logic to the Navigator.

Advanced extension challenges

- **Emergency stop that actually stops** plus a way to restart without reloading.
- **Six-key panel** with diagonal movement.
- **Message cascade:** one key triggers a broadcast that triggers a second.
- **HUD with three status sprites**, each subscribed to its own message.
- **Name the pattern:** two sentences on why "a signal everyone can hear" helps when four people build one project.
- **Robot tables only:** add a fifth button that broadcasts `slow`, and wire it to a lower speed. Two messages, one robot, no new concepts.

Robot-table adjustments

- One student drives at a time, seated, device held with two hands. Standing drivers turn too sharply.

- Keep the robot on the taped area. If it leaves, switch it off, put it back, continue — do not chase it.
- If a robot will not pair within 60 seconds, swap to the spare and note it. Do not let a pairing failure become the lesson.

WEEK 5

Start, Stop, and State

1. Lesson Info Block

Topic	Events that begin vs end; representing state with costumes
Duration	40 minutes
Keywords	state · mode · start · halt · resume · indicator · simultaneous scripts
Mastery statement	*"A program can be in different conditions, and I can show which one."*
Materials	Devices · printed studio task · `P1_05_PatrolKit` pre-loaded (sprite with three costumes: **grey IDLE / green ACTIVE / red HALT**) · exit tickets

2. Teacher Preparation Checklist

- [ ] Build `P1_05_PatrolKit`: one sprite, three costumes named exactly `idle`, `active`, `halt`. Write those words on the board.
- [ ] Verify `stop other scripts in sprite` exists in Padma. If not, use the broadcast-and-costume substitute and note it.
- [ ] Prepare a broken demo: a patrol that cannot be stopped.
- [ ] Print the studio task with the **state diagram** pre-drawn and three arrows to complete.
- [ ] Write on the board: **IDLE → (tap) → ACTIVE → (S) → HALT → (G) → ACTIVE**

3. Step-by-Step Instructional Guide

WARM-UP — "FREEZE" (5 MIN)

> **Script:** "When I say *dance*, you dance. When I say *freeze*, you stop — completely, mid-move. Ready? … Dance!"

Fifteen seconds. "Freeze!" Five counts. "Dance!"

> **Script:** "Hands up if you *knew* you were frozen. … You knew because something remembered the last command. That's **state** — the condition you're in right now. A computer needs it too. Without it, 'stop' and 'go' are just noises."

Then: "What if I said *freeze* and there was no instruction to unfreeze? You'd stand there forever. Today is about always leaving a way back."

GUIDED INSTRUCTION — THREE MODES, ONE SPRITE (10 MIN)

Step 1 (3 min). `when [sprite] clicked → switch costume to [active] → forever [ move [5] steps, turn [3] degrees ]`.

> **Script:** "One tap and it does its job forever. Now I need to stop it — and I need to know whether it's running without looking at it. That's what the costume is for."

Step 2 (4 min). `when [s] key pressed → switch costume to [halt] → stop other scripts in sprite`.

> **Script:** "Read it carefully. Not `stop all`, not `stop this script` — `stop **other** scripts in **this sprite**`. With `stop all` it would kill the script I'm standing in and the costume change might not finish."

Step 3 (3 min). Pressing G returns to base, turns green, patrols again. Build it on the board with the class contributing blocks.

> **Script:** "Three conditions: ready, working, stopped. Each has a name and a colour. That's a **state machine** — and it's exactly what the robot needs next term."

Draw the three-circle diagram; students copy it.

Step 4 — the loop-back arrow (3 min). Under the state diagram, draw the *flowchart* of the patrol script:

oval `start` → **parallelogram** `switch to active` → **rectangle** `move 5` → **rectangle** `turn 3` → **arrow back to** `move 5`.

> **Script:** "Two different pictures, two different questions. The circles tell you **what can happen**. The flowchart tells you **what the program does**, in order. One is a map of states; the other is a map of steps."

> **Script:** "Look at that arrow going back up. That is your `forever` loop, drawn. In a flowchart a loop is not a special shape — it is simply an arrow that goes backwards. That's all a loop ever was."

> **Script:** "Next week you'll need a loop inside a loop. You'll draw it, and the drawing will have an arrow going back inside a bigger arrow going back. Draw it first and you won't get lost in the blocks."

INDEPENDENT PRACTICE — STUDIO TASK: PATROL PROTOCOL (20 MIN)

1. Grey (`idle`) at start, **stationary**
2. Tap → green (`active`) → patrols forever
3. **S** → red (`halt`) → movement stops *and* it says `"Halted."` for 1 second
4. **G** → returns to base → green → patrols again
5. The whole cycle must work **three times without reloading**

Save as `P1_05_Patrol`.

Task 0 (4 min, before you build): draw the flowchart of your patrol script. The teacher initials it.

The test that matters: a student from another team starts, halts and restarts it *without asking how*. Write that on the card.

DEBUGGING & DISCUSSION — "WHY WON'T IT STOP?" (5 MIN)

Show the broken demo (no halt script). Then the subtler failure:

> **Script:** "The sprite stops moving but keeps sliding. Why? There were **two scripts running on the same sprite**, and `stop other scripts` only stopped the one in the same chain. Count the hats. Two forever-loops means two problems."

Second: **restart leaves it in the wrong place.** "State isn't just the costume — it includes *where* and *which way*."

Exit ticket: "*Name the three states and one thing the sprite does in each.*"

4. Differentiations

Supporting struggling students

- Two states only: idle and active, toggled by one key. Halt is extension.
- `P1_05_Starter` with the start script complete and an empty `when [s]` hat waiting.
- Give a **physical prompt card**: three coloured squares matching the costumes, with the trigger written under each.

- Confirm at minute 20 that halt works; don't let them leave without it.

Advanced extension challenges

- **Four-state machine** with `EMERGENCY` requiring a two-tap reset.
- **Random patrol** — then argue whether randomness is good for a real robot.
- **Two-sprite drill:** the second halts when the first halts, via broadcast.
- **Status panel** reading the state as text, not costume.
- **Write the test plan:** three cases with expected results.

WEEK 6

Loops: The Copy-Paste Is a Symptom

1. Lesson Info Block

Topic	Counted repetition; recognising and replacing duplication
Duration	40 minutes
Keywords	loop · repetition · count · duplicate · refactor · efficiency
Mastery statement	*"If something repeats, say it once."*
Materials	Devices · printed studio task · **A3 "40-block square" poster** up before students enter · 5 large instruction cards for the human-loop demo · **4 KanKaung robots, paired and charged** (the other half of the tables) · exit tickets

> **Robot week (second of two).** Tables 5–8 drive today. Tables 1–4 drove in Week 4.

2. Teacher Preparation Checklist

- [ ] Tape the 40-block poster up before the lesson. Do not explain it — let them look.
- [ ] Print 5 large cards: four identical `move 100 / turn 90`, one `repeat 4 times`.
- [ ] Prepare `P1_DEMO_6`: 40-block program on one side, 2-block loop version on the other.
- [ ] Pre-load `P1_06_Starter` for support: the long-form square, ready to be refactored.
- [ ] Reserve a board section for: **"What is identical? How many times?"**

3. Step-by-Step Instructional Guide

WARM-UP — "THE HUMAN LOOP" (5 MIN)

Five students to the front. Four get identical `move 100 / turn 90` cards; the fifth gets `repeat 4 times`.

> **Script:** "You four, read your card when I point. … Move, turn. Move, turn. Move, turn. Move, turn. Now — you, with the big card."

> **Script:** "How many people does the second version need? One. How many instructions do we hear? Two, instead of eight. Both draw exactly the same square. One is a **program**. The other is a **transcript**."

Point at the poster.

> **Script:** "That is what happens when you never learn this. Forty blocks. It works. It's also unreadable, unfixable, and if I ask you to change the size you'll be editing forty numbers."

GUIDED INSTRUCTION — FIND IT, WRAP IT (10 MIN)

Step 1 (2 min). Write the question and have the class read it aloud:

> **"What is identical? How many times does it happen?"**

Step 2 (5 min). Eight-block square: side one is `move 100 / turn 90`; side two is identical; four times.

Drag `repeat [4]` and drop the two blocks **inside** it.

> **Script:** "Inside. Not above, not below. The indented blocks run four times. The rest run once. **The indentation is the meaning.**"

Step 3 (3 min). `repeat [36]` with `move [10]` and `turn [10]`. A circle appears.

> **Script:** "A circle from two blocks. Every point is a straight line too short to see the corner. Now change 36 to 4 and 10 to 90 — you're back to the square. **The loop count and the turn angle are the same idea.**"

INDEPENDENT PRACTICE — STUDIO TASK: LOOP LADDER (20 MIN)

RUNG	TASK	CONSTRAINT
1	Rebuild the Week 2 square with exactly **3 blocks**	Identical output
2	Hexagon using one loop	6 sides
3	12-sided shape — circle or polygon?	Discuss
4	Three separate squares using **one** loop	Needs `pen up` inside or around
5	Eight small squares, each rotated 45°	Loop inside a loop
R	**Robot tables:** drive KanKaung while a button is held, and stop the moment it is released	`repeat until`

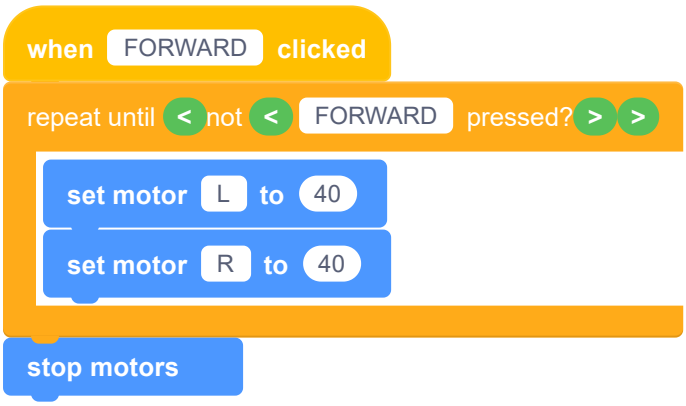
Save as `P1_06_Loops`.

Rung 4 discriminates. Students who only copy patterns draw three squares joined by a visible line; the fix — lifting the pen to travel — is a real insight.

Rung 5 is deliberately hard. Say out loud: "Struggling here is expected and correct."

Rung R — the robot loop (8 minutes, tables 5–8 today)

The program from Week 4 drives until you press stop. Now make it drive **only while a button is held** — a loop with an exit condition.



> **Script:** "Read the condition out loud. *Repeat until the button is no longer pressed.* The loop runs while it is held, and stops the instant it isn't. That's not a count — it's a **condition**. Next term you'll use the same shape for a sensor instead of a finger."

The test: hold the button and walk away. The robot must stop when the finger lifts. If it keeps going, either the loop is wrong or the command is only being sent once.

> **Script (after):** "In Week 4 the robot ran until you told it to stop. Now it stops when you stop meaning it. That's the first time a loop in this course has been responsible for something physical."

Rotation note. Tables 5–8 drive today; tables 1–4 did so in Week 4. Rungs 4 and 5 become the non-driving tables' task.

DEBUGGING & DISCUSSION — "LOOPS THAT DON'T LOOP" (5 MIN)

Draw three broken loops on the board — faster and clearer than a device:

1. `repeat 4` containing only `move 100` → a straight line. "The loop repeats, but nothing changes between repetitions."
2. `turn 90` **outside** the repeat → one turn at the end. "Inside means *every time*. Outside means *once*."
3. `repeat 0` → nothing, with no warning. "Zero is legal. Check your counts."

Exit ticket: "*Write the two blocks inside a loop that draws a hexagon.*"

4. Differentiations

Supporting struggling students

- Rungs 1 and 2 with `P1_06_Starter` — refactoring is a *reduction* task, far less intimidating than construction.
- Physical loop card: a rectangle with a notch where two blocks slide in. Put cards in the notch before touching the screen.
- Drop the "3 blocks" constraint; accept "fewer than before."
- One sentence only: "What repeats? How many times?"

Advanced extension challenges

- **Loop budget:** a 20-pointed star in under 8 blocks; compare with a partner.
- **Nested flower** with both loops named ("petal loop", "flower loop").
- **Refactor audit:** count blocks on the poster, count their own, write one sentence about what changed for a *reader*.
- **Teach it:** a 60-second explanation of loops for a Grade 3 class.

WEEK 7

Nested Loops: The Lotus Wall

1. Lesson Info Block

Topic	A loop inside a loop; 2D patterns; pattern as cultural design source
Duration	40 minutes
Keywords	nested loop · outer loop · inner loop · tile · motif · grid · repetition of repetition
Mastery statement	*"A loop can repeat a loop."*
Materials	Devices · printed studio task · **3-4 images of lotus motifs** (woven, carved, painted) · blank grid paper (4 per team) · floor grid 3×3 with 9 identical objects · exit tickets

2. Teacher Preparation Checklist

- [ ] Bring a **real cultural object** if you can — a woven mat, a lacquerware piece, a carved panel. Physical artefacts beat photographs, and this lesson is Padma's namesake moment.
- [ ] Print 3–4 lotus motif images, each with a clear repeating unit and a repeating arrangement.
- [ ] Tape a 3×3 grid on the floor and place 9 identical objects in it — the physical nested loop, understood in one minute.
- [ ] Prepare `P1_DEMO_7` with a 3×3 grid from nested loops, plus a broken version missing the reset step.
- [ ] Pre-load `P1_07_TileTemplate` for students who need the drawing done for them.

3. Step-by-Step Instructional Guide

WARM-UP — "FIND THE UNIT" (5 MIN)

Hold up the object or image.

> **Script:** "Look at this. Complicated — dozens of shapes. But it isn't made of dozens of ideas. It's made of **one idea, repeated**. Who can find the smallest piece that repeats?"

Then ask how the motifs are arranged — in a row, or in a grid.

> **Script:** "So there are two repetitions. The **unit** repeats inside a row. The **row** repeats down the cloth. Say it back: repetition of repetition. That's a **nested loop**, and humans have been programming cloth this way for two thousand years."

Walk to the floor grid.

> **Script:** "Nine objects. Did I place nine things? No. I placed one row three times. That's the outer loop. Each row is one thing three times. That's the inner loop."

GUIDED INSTRUCTION — OUTER AND INNER (10 MIN)

Step 1 (3 min). One tile: `repeat 4 [ move 20, turn 90 ]`.

Step 2 (2 min). Copy it three times by hand. "It works. And it's ugly. What repeats? The tile. How many times? Three, in a row."

Step 3 (3 min). Wrap in `repeat [3]` with `move [40]` after each tile.

> **Script:** "Now we have a row. But a wall is three rows. And here's the trick — the whole row repeats too. So we put a loop around *the loop*."

Step 4 (2 min). Wrap the row in another `repeat [3]`, adding "go back to the start and drop down."

> **Script:** "**Outer** loop: which row. **Inner** loop: which tile. Between rows I jump back to the left and move down. That jump is what most people forget — and it's the bug you'll meet in five minutes."

INDEPENDENT PRACTICE — STUDIO TASK: DESIGN A LOTUS TILE · CARD 3 (20 MIN)

Part 1 (6 min). On grid paper, design a motif of 4–9 cells. Recognisable, and *yours*.

Part 2 (6 min). Draw it with the pen.

Part 3 (8 min). Tile it 3×3 with a nested loop.

Requirements: at least one loop inside another; equal gaps; fills ≥ 2/3 of the stage; one notebook sentence — "*My outer loop does ___ and my inner loop does ___*."

Save as `P1_07_Lotus`.

DEBUGGING & DISCUSSION — "THE DRIFTING WALL" (5 MIN)

Show the broken demo: no reset between rows, tiles march diagonally off-stage.

> **Script:** "It's not random. It's *drifting* — one step wrong, three times. Whenever a pattern drifts instead of scattering, you're looking at a missing reset."

Second: `pen down` left on during the jump, producing a diagonal smear.

> **Script:** "Two explanations, one picture. Is the pen down, or is the position wrong? **Change one thing and re-run.** That's how you tell."

Exit ticket: *"In a 3×3 wall, how many times does the inner loop run in total?"* (9)

4. Differentiations

Supporting struggling students

- `P1_07_TileTemplate`: motif pre-drawn; they build only the nested loop. Isolates the new concept from drawing skill.
- Reduce to a **2×2 wall**.
- Walk the floor grid: "You are the sprite. Draw tile one. Now what do you do before tile two?"
- Give the reset step as a printed card in words they translate.
- Accept a single row plus a written explanation of what the outer loop *would* do.

Advanced extension challenges

- **Brick offset** — shift every second row by half a tile.
- **Two motifs** alternating in a checkerboard.
- **Border** around the wall with a separate loop.
- **5×5 with a colour change per row.**
- **Documentation challenge:** instructions precise enough for another team to reproduce it with no picture.

WEEK 8

Composition: Three Constructs at Once

1. Lesson Info Block

Topic	Composing sequence, events and loops into one responsive system
Duration	40 minutes
Keywords	composition · integration · interruption · simultaneous scripts · conflict · autonomy
Mastery statement	*"A loop does the work; an event changes the plan."*
Materials	Devices · printed studio task · 3-colour sticky notes · `P1_08_PatrolKit` · exit tickets

> **No hardware this week.** The robot visited half the tables in Week 4 and the other half in Week 6. Every team has driven it once. Today they build the program it will eventually run itself.

2. Teacher Preparation Checklist

- [ ] Prepare `P1_DEMO_8` with two scripts (patrol loop + edge reaction) and a broken variant missing one block.
- [ ] Pre-load `P1_08_Starter` for support (patrol loop complete, reaction script empty).
- [ ] Print 8 sets of 3-colour sticky notes: **red = event, yellow = sequence, green = loop**.
- [ ] Write on the board: **KanKaung's job is this program, running without us.**
- [ ] Have the robot on your desk, switched off. Do not use it today — its presence is the point.

3. Step-by-Step Instructional Guide

WARM-UP — "LABEL THE MACHINE" (5 MIN)

Project a ~15-block program containing a hat, a short sequence and a loop.

> **Script:** "This program has three kinds of thing in it, and you know all three. Red goes on anything that **starts** something. Yellow on anything that runs **once, in order**. Green on anything that **repeats**."

90 seconds. One team places their notes on the projected image.

> **Script:** "Every program you build for the rest of the year is these three things, combined. You've built each one separately. Today you build the combination — because a machine that follows a line has to do all three at the same time."

GUIDED INSTRUCTION — THE INTERRUPTION PROBLEM (10 MIN)

Step 1 (3 min). Build `when [sprite] clicked → point in direction [90] → forever [ move [6] steps, wait [0.05] seconds ]`. It patrols forever.

Step 2 (4 min). Add a separate event on the same sprite: `when [space] key pressed → say [Sample detected!] for [1] second`.

> **Script:** "The sprite is patrolling, inside a `forever` loop. I press space. Does it stop patrolling? … No! It does both. Why?"

Let them work it out: **the loop is in one script, the key press in another.** Two separate programs sharing a sprite.

> **Script:** "This is the biggest idea of the week. A **script** is a program. A sprite can run several at once. That's how you make something that keeps doing its job *and* responds to you."

Step 3 (3 min). Point at the robot on your desk.

> **Script:** "That machine has to do exactly this, four hundred times a second, with nobody watching. One loop reading the sensors. One event answering the stop button. Both at once. Today's program is its skeleton — you'll give it a body in Term 3."

INDEPENDENT PRACTICE — STUDIO TASK: REACTION PATROL (20 MIN)

#	REQUIREMENT	CONCEPT
1	Sprite patrols continuously from one tap	loop
2	When it touches the left or right edge it says `"Obstacle."` and plays a sound	event ↔ loop coexistence
3	Pressing **space** makes a separate status sprite report `"Nominal."`	broadcast
4	Pressing **S** halts the patrol *and* updates the status to `"Stopped."`	state
5	All four work **simultaneously** for 60 seconds without reloading	integration

Save as `P1_08_ReactionPatrol`.

Requirement 5 is the assessment. Make them run a 60-second test with a visible timer and record the result.

DEBUGGING & DISCUSSION — "TWO SCRIPTS, ONE SPRITE" (5 MIN)

Project the failure: a patrol loop and a key-press script both moving the sprite. It jitters.

> **Script:** "Both scripts are telling the sprite where to be. Neither is wrong. They're both right at the same time. This is a **conflict**, and it's not a bug in a block — it's a design problem."

Then connect it forward:

> **Script:** "In Term 2 you'll write the rules that decide which script wins. In Term 3, on a machine that weighs four kilos, a conflict isn't a jitter — it's a crash."

Exit ticket: "*Why does the sprite keep patrolling while it says 'Obstacle'?*"

4. Differentiations

Supporting struggling students

- `P1_08_Starter`: patrol loop complete, reaction hat placed. They add three blocks.
- Requirements 1 and 2 only.
- Provide the exact text strings on a card so language isn't the barrier.
- If two scripts conflict, help them delete one and rebuild it separately — conflict resolution is Week 9.

Advanced extension challenges

- **Three-state patrol:** IDLE / ACTIVE / ALERT, with ALERT triggered by an edge touch.
- **Sweep scan:** an arc sweep reporting what it "sees" at each step.
- **Sound design:** three sounds for three events, plus a written justification of why sound matters on a machine with no screen.
- **Draft KanKaung's first rule** on paper: what should it do when a sensor sees nothing? Keep it — you'll check it in Term 3.

WEEK 9

Debugging Studio: The Six-Step Protocol

1. Lesson Info Block

Topic	Systematic fault-finding: predict → run → observe → isolate → fix → re-test
Duration	40 minutes
Keywords	bug · fault · isolate · reproduce · hypothesis · regression · test case
Mastery statement	*"I change one thing at a time, and I prove the fix."*
Materials	Devices · **6 pre-loaded broken projects** `P1_Bug1` – `P1_Bug6` · Bug Log worksheet · **hint cards** (2 per bug, sealed) · A3 poster for the Bug Wall · exit tickets

2. Teacher Preparation Checklist

- [ ] **Build and test all six broken projects yourself.** A bug that won't reproduce wastes a team's whole lesson.
- [ ] Verify each fails **deterministically**.

- [ ] Two hint cards per bug in sealed envelopes: hint 1 = "look in this script", hint 2 = "the fault is this block". Using a hint costs a rubric level, not a grade.
- [ ] Print the six-step protocol **large** and tape it to the wall — it stays all year.
- [ ] Arrange six stations, one device cluster each, so teams rotate.

The six-bug bank

#	BUG	SYMPTOM	LESSON
1	No hat block, bare `move`	Nothing happens	Nothing runs without a trigger
2	Triangle drawn with `turn 60`	Won't close	Wrong parameter, not wrong logic
3	`repeat 4` above the blocks instead of around	Only the last block repeats	Nesting is positional
4	`pen down` missing at the start	First side invisible	Sequence matters
5	`broadcast [Arrived]` / `when I receive [arrived]`	Silent failure	The join is a place to look
6	Two loops changing the same position	Jitter	Sometimes the fix is deleting

3. Step-by-Step Instructional Guide

WARM-UP — "SPOT THE DIFFERENCE" (5 MIN)

Project two nearly identical programs; one works, one doesn't. 60 seconds, countdown.

> **Script:** "Sixty seconds. Do not run them. Read them. Find the difference. Go."

Take answers without judgement.

> **Script:** "Three guesses, one right. But notice what you did — you didn't guess randomly, you *compared*. You looked for what was different. That's the whole skill, and today you get a method so you don't need luck."

GUIDED INSTRUCTION — THE SIX STEPS, DEMONSTRATED (10 MIN)

Teach it by using it. Project Bug 4.

STEP	DO	SAY
1. Predict	Say what should happen	"This should draw a closed square."

STEP	DO	SAY
2. Run	Run once	"Here's what actually happened."
3. Observe	Say exactly what's wrong	"The left side is missing. Everything else is right."
4. Isolate	Shrink the search	"Is it drawing, order, or pen? I'll add `pen down` at the top and re-run."
5. Fix	Change **one** thing	"One block. Nothing else."
6. Re-test	Run again, and twice	"Fixed. Now again — the second run is where most fixes fail."

> **Script:** "Step 3 is the one you'll want to skip. '*It doesn't work*' is not an observation. '*The third side is too short*' is. You cannot find a fault you haven't described."

Then do **Bug 1** with a student driving.

INDEPENDENT PRACTICE — THE SIX STATIONS (20 MIN)

Teams rotate every **3 minutes**. **They do not have to fix every bug** — the goal is six completed rows in the log.

Log columns: bug · prediction · what happened · how isolated · one-line fix · hint used?

Rules: you may read the code; you may **not** delete everything and start over; open a hint only after **two** failed attempts; every fix must be re-tested **twice**.

DEBUGGING & DISCUSSION — BUILD THE BUG WALL (5 MIN)

Each team writes its best insight on the wall poster under the six numbered boxes. Then name the three patterns:

1. **"Nothing happens"** — missing trigger or mismatched name.
2. **"It's almost right"** — parameter or order.
3. **"It fights itself"** — two scripts doing the same job.

> **Script:** "These six bugs aren't school bugs. They're the six you'll meet in your project next week, and next term on the robot. The wall stays up all year."

Exit ticket: "*Write the six steps from memory.*"

4. Differentiations

Supporting struggling students

- Bugs 1, 3 and 4 only — trigger, nesting, sequence.
- Station 1 has a worked example printed beside the device.

- Hint cards are **free** for these students.
- Pair with a strong Recorder who writes the log while they hunt.
- Give the six steps as a checkbox strip they tick.

Advanced extension challenges

- **Bug exchange:** write a new deliberate bug into a copy of the Week 8 project and swap with another team.
- **Intermittent bug:** a fault that only appears on the second run (state not reset) — the hardest category, and a preview of Term 2.
- **Time trial:** all six in 12 minutes, then compare methods.
- **Write the class debug card** in their own words, for a later class to use.

WEEK 10

Build Sprint: The Mission Console

1. Lesson Info Block

Topic	Independent construction of the Term 1 project
Duration	40 minutes
Keywords	plan · decompose · waypoint · tolerance · checkpoint · definition of done
Mastery statement	*"I can plan it, build it, and prove it runs alone."*
Materials	Devices · **Mission Brief** (one per team) · **Planning Sheet** (one per team, mandatory) · rubric extract (criteria A and B) · visible timer · spare device

2. Teacher Preparation Checklist

- [ ] Print Mission Brief and Planning Sheet — **no device opens without a signed plan**.
- [ ] Print the rubric extract so students can read the standard they're judged against.
- [ ] Set a **visible countdown timer**.
- [ ] Prepare `P1_10_Starter` for support teams: sprite, background, start marker placed.
- [ ] Decide three checkpoints: minute 12 (plan signed), minute 22 (half built), minute 36 (complete or blockage documented).

3. Step-by-Step Instructional Guide

WARM-UP — READ THE MISSION (5 MIN)

Read the brief aloud, in role, once. Then stop.

> **Brief:** "*Your console must run a four-waypoint survey with no operator present. One tap starts it. Every waypoint is visited in order. When the route is complete, the console reports status and stops. If it stops before the end, it must say why.*"

> **Script:** "Nothing in that brief mentioned a loop or an event. But you already know what it needs. What does it need?"

Collect two or three answers; write them as a list: *a trigger · repeated travel · a way to stop · a report.*

> **Script:** "That list is a **decomposition**. It's the plan. Today we don't start with blocks — we start with paper."

GUIDED INSTRUCTION — MODEL THE PLAN, NOT THE CODE (10 MIN)

Step 1 (4 min). Fill a Planning Sheet on the board for the first two waypoints, narrating the thinking.

> **Script:** "Waypoint 1 is at (2,3). I could go across then up, or up then across. Which one misses the blocked square? Let me trace both with my finger. … Across then up crosses it. Decision made before any coding."

Step 2 (3 min). Write the definition of done with the class:

- Runs from one tap
- Visits all four waypoints in order
- Ends within one square of the target
- Reports status at the end
- Runs twice without reloading

> **Script:** "Point five is the one that catches people. Run it twice. If the second run differs, you have a state bug."

Step 3 (3 min). Show the rubric extract.

> **Script:** "*Design and decomposition* means your plan and your code agree. *Correctness and debugging* means it runs alone and you can name what you fixed. Neither is about how clever the code looks."

INDEPENDENT PRACTICE — SPRINT (20 MIN)

Rules: plan first, signed before the device opens; one tap to start; log every bug as you fix it; 60-second status check at minute 12 and minute 22.

TIME	CHECKPOINT	IF RED
12 min	Plan signed, first waypoint coded	Switch to the 3-waypoint route
22 min	Reaches waypoint 3	Drop the last two — a short route that works beats a long one that doesn't
36 min	Complete, or blockage documented	Document and continue next lesson

Teacher circulation rule: **question ladder only.** No touching devices, no pointing at blocks.

1. "What did you expect?" 2. "What actually happened?" 3. "Which part of the plan does that?" 4. "What's the smallest test that would tell you?" 5. "What did you learn in the last two minutes?"

Save as `P1_10_Console`**.** Early finishers start the status report.

DEBUGGING & DISCUSSION — "THE BROKEN ATTEMPT" (5 MIN)

Ask for **one volunteer** to demonstrate a route that does *not* work. Make this prestigious: "Who has an interesting failure?"

Run it. The class diagnoses using the six steps — you only ask "which step are we on?"

> **Script:** "Ten weeks ago you couldn't make a sprite talk. Today you have a system that plans a route and reports on itself. Next lesson you finish it, and then — the robot."

Exit ticket: "*One thing my console does well. One thing it doesn't do yet.*"

4. Differentiations

Supporting struggling students

- **Route A** from the start — 3 waypoints, no obstacles. Present it as "the short mission."
- `P1_10_Starter` with the sprite, background, start marker and first `go to` pre-filled.
- Drop "runs twice" from the definition of done.

- Pre-fill waypoint coordinates; the only decision is the order.
- Assign a strong Navigator to trace the route on paper.

Advanced extension challenges

- **Obstacle reroute** mid-run.
- **Status report** as a separate sprite using broadcast.
- **Survey pattern** drawn over the stage as the route runs (nested loop, reusing Week 7).
- **Write the operator card** in under 40 words.
- **Mentor role:** finished teams become "mission control" for a struggling team — may answer questions, may not touch the device.

Term 1 at a glance

WK	LESSON	NEW CONCEPT	CUMULATIVE BUILD	ROBOT
1	Meet Padma, Meet the Robot	Hat blocks, events	First script	on the desk
2	Precision Drawing	Sequence, parameters	Closed shapes	—
3	The Stage as a Map	Coordinates, heading	Planned route	—
4	Mission Control	Broadcast	**Remote control for KanKaung**	**tables 1–4**
5	Start, Stop, and State	State machine	Patrol protocol	—
6	Loops	`repeat N`, refactoring	**Hold-to-drive loop**	**tables 5–8**
7	Nested Loops	Nested loops	Lotus wall	—
8	Composition	Three constructs at once	Reaction patrol	—
9	Debugging Studio	Six-step protocol	Bug wall	—
10	Build Sprint	Decomposition	Console, half built	—

Ten things every student must be able to do by Week 12

1. Make a sprite respond to a tap and a key press.
2. Draw a closed shape using exact angles.
3. Place a sprite at a coordinate and explain the origin.
4. Send a message another sprite reacts to.
5. **Drive KanKaung from four sprite buttons and stop it deliberately.**

6. Explain the difference between an event and a loop.
7. Replace duplicated blocks with a loop and say why.
8. Put a loop inside a loop and explain which does what.
9. Follow a six-step debugging method.
10. Plan a route on paper before building it.

Before Term 2 begins

- [ ] Every project saved under `P1_NN_Name` and copied to the class SD card.
- [ ] Bug Wall poster still up — Term 2 needs it.
- [ ] Notebooks collected, one page initialled per week.
- [ ] **Robots charged, paired, and wheels tested** — Term 2 Week 1 assumes working hardware.
- [ ] Exit tickets sorted into three piles; the "confused" pile drives Term 2 Week 1.
- [ ] One **golden build** of the Mission Console sealed as a moderation reference.

Term 1 pack v1.0 · Padma Robotics Lab · Awaken Innovation Lab