PADMA ROBOTICS PROGRAMME — TERM 2 LESSON PLANS

Term 2 · Rules · Weeks 1–10 · Grades 6–8

Format: 10 × 40-minute lessons · **Devices:** 1 per 4 students **Hardware:** 4 KanKaung robots, used in **Week 5** and **Week 7** (half the tables each)

> **Prerequisite:** Term 1. The lesson shape, roles, debug card and file naming are unchanged.
>
> **What this term is.** Term 1 taught students to make things obey. Term 2 teaches them to make things **decide** — and to write those decisions down as rules. Two robot sessions put the rules into a real machine; everything else happens on screen.
>
> **What this term is not.** This is not the robot term. Term 3 is. Everything built here gets executed there.

The through-line

Every week answers one question that KanKaung will later ask:

WEEK	THE QUESTION	WHERE IT LANDS IN TERM 3
1	Can a machine test this?	Which sensor readings are usable
2	What happens when the test fails?	The `else` branch that stops the robot
3	What if there are four answers, not two?	**The four-state table**
4	How does a machine remember?	Speed, correction, blocked
5	How do I tune without rewriting?	Trim, calibration
6	What if the machine has moods?	PATROL / AVOID / STOP
7	How do I write it once and use it twice?	The follower routine
8	How do I make a routine general?	A sweep across many angles
9	How do two teams build one machine?	Message contracts
10	How do I know it's right?	Boundary testing

Say this in Week 1 and again in Week 10: *"Nothing you build this term runs on the robot. Everything you build this term is the robot."*

Blocks this term

BLOCK	PURPOSE
`if <> then` / `if <> then … else`	Selection
`and` · `or` · `not`	Compound conditions
`set [x] to` · `change [x] by` · `show variable`	Variables
`define [name]` · custom block with parameters	Functions
`broadcast` / `when I receive`	Messages
`set motor [L] to` · `drive (l) (r)`	Robot only, Weeks 5 and 7

File naming: `P2_01_Decide`, `P2_02_Else`, … `P2_10_Test`.

Standard materials

Devices · printed challenge card · debug card · notebook · exit tickets · **4 robots, paired and charged** (Weeks 5 and 7 only) · masking tape.

WEEK 1

The Decision Problem

1. Lesson Info Block

Topic	What a condition is; statements a machine can test; **combining two tests — AND, OR, NOT**
Duration	40 minutes
Keywords	condition · true · false · test · measurable · gate · truth table · input · output · trigger vs check
Mastery statement	*"A machine can only decide about things it can measure."*

Materials	Devices · 32 statement cards (4 sets of 8) · sorting mats (TRUE / FALSE / CAN'T TELL) · **gate sheet (one per table)** · `P2_01_Starter` · exit tickets

2. Teacher Preparation Checklist

- [ ] Print and cut 32 statement cards, 4 sets of 8. Mix always-true, currently-true, currently-false, and undecidable:
  - "The door is shut." · "It is Tuesday." · "There are twelve chairs in this room."
  - "The crew is happy." · "This room is beautiful." · "The robot is broken."
  - "The battery is above 40%." · "The wall is closer than two metres."
- [ ] Write on the board: **CAN MEASURE / CANNOT MEASURE.**
- [ ] Print one **gate sheet** per table: the AND, OR and NOT symbols, and three blank four-row truth tables.
- [ ] Prepare a demo that fails without a decision: a sprite that walks off the stage edge.
- [ ] Put the robot on your desk. Do not switch it on.

3. Step-by-Step Instructional Guide

WARM-UP — "TRUE, FALSE, OR CAN'T TELL" (5 MIN)

Hand each team eight cards and three labels.

> **Script:** "These are statements. Some are true right now. Some are false. Some — a machine could never work out. Sort them. Two minutes. Argue."

At the end, ask where two specific cards went.

> **Script:** "You couldn't place 'the crew is happy', could you? There's nothing to measure. But 'the battery is above 40%' — that has a number, and a number can be checked. **A machine can only decide about things it can measure.**"

GUIDED INSTRUCTION — WHY IT FELL OFF (14 MIN)

Step 1 (3 min). Run the demo. The sprite patrols and vanishes off the edge.

> **Script:** "It works perfectly. And it drives off the edge of the world. Why?"

Collect answers. Someone will say "because you told it to."

> **Script:** "Exactly. It has an instruction, not a decision. Look at every block and ask: does any of them ask a question? … None. This program is blind."

Step 2 (4 min). Add one block inside the loop: `if <touching edge?> then [ turn 180 degrees ]`.

> **Script:** "One block turned this from a machine that obeys into a machine that decides. Read it out loud. That's a **condition**: a test with two outcomes."

Step 3 (3 min). The three parts, on the board:

TEST (is it touching?) → **DECISION** (true / false) → **ACTION** (turn, or don't).

> **Script:** "Every condition this term is those three parts. If you can't name the test, you don't have a condition — you have a guess."

Point at the robot.

> **Script:** "In nine weeks that machine will be making four of these decisions, four hundred times a second, with nobody watching. Today you write your first one."

Step 4 — the gates (4 min). Draw the three symbols on the board:

GATE	SYMBOL	OUTPUT TRUE WHEN	SAY IT AS
AND	D-shape, two inputs	both inputs are TRUE	"both"
OR	curved back, two inputs	at least one input is TRUE	"either"
NOT	triangle with a dot	its single input is FALSE	"not"

Fill the AND truth table with the class:

A	B	A AND B
TRUE	TRUE	**TRUE**
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

> **Script:** "A gate takes TRUE or FALSE in, and gives TRUE or FALSE out. That is all it is. **AND is strict** — both must be true. **OR is generous** — one is enough. **NOT flips one.** Every `if` you write is one of these gates with an action bolted on."

> **Script:** "The table has one row for every combination of inputs. Two inputs means four rows — always. You will meet this exact table in Week 3, when the inputs are KanKaung's two sensors."

60-second pair task. "Copy the table, change the heading to OR, fill it in, then compare with your partner."

INDEPENDENT PRACTICE — STUDIO TASK: TERMINAL REFRESH (17 MIN)

TASK	REQUIREMENT	TIME
A	Sprite patrols and reverses at either edge	5 min
B	It says `"Obstacle."` for one second at each reversal	4 min
C	Write five statements your robot **could** test, the sensor each needs, then combine two of them with AND and with OR — four-row truth table for each	6 min
D	Notebook: name the three parts of your condition	2 min

Save as `P2_01_Decide`.

Task C is the bridge to Term 3. Expect answers like "is something in front of me?" (ultrasonic), "am I on the line?" (IR), "is it bright?" (LDR).

DEBUGGING & DISCUSSION — "CHECKED ONCE OR ALWAYS?" (4 MIN)

Show the `if` placed **outside** the loop.

> **Script:** "It checks once, at the start. At the start it isn't touching the edge, so it does nothing, ever. The program isn't wrong — it answered correctly. It just never asked again."

Then the inverse.

> **Script:** "This is the difference between a guard who checks the door once at 6 a.m. and one who watches all night. **Position inside the loop is the whole difference.**"

Exit ticket: "*Write one statement a machine can test, and one it cannot.*" and "*Which gate is TRUE only when both inputs are TRUE?*"

4. Differentiations

Supporting struggling students

- Give `P2_01_Starter` with the `if` block pre-placed and the action empty.
- Reduce Task C to two statements.
- Say the test out loud in English first, then hunt for the block.

Advanced extension challenges

- Two-condition patrol: reverse at an edge *and* react differently at a corner.
- Find five statements in the room a machine could test, and name the sensor for each.
- Write the manual paragraph for a system that does not exist yet.

WEEK 2

If / Else

1. Lesson Info Block

Topic	Branches and comparison operators; what happens when the test fails
Duration	40 minutes
Keywords	branch · else · comparison · greater · less · equal · test case
Mastery statement	*"Every test has two outcomes, and I have to build a road for both."*
Materials	Devices · gatekeeper cards (6 valid, 4 invalid per team) · yellow and red cards · `P2_02_Airlock` · exit tickets

2. Teacher Preparation Checklist

- [ ] Print gatekeeper cards — 10 per team, 6 marked VALID (green dot on the back), 4 INVALID (red dot). Dots on the back only.
- [ ] Build `P2_02_Airlock`: a door sprite, a pressure display, two costumes, a warning sound.
- [ ] Write on the board: **IF the test passes → do this. IF it fails → then what?**
- [ ] Prepare the broken demo: an `if-else` where both branches do the same thing.

3. Step-by-Step Instructional Guide

WARM-UP — "THE GATEKEEPER" (5 MIN)

Stand at the door with the yellow and red cards.

> **Script:** "I am the airlock. You have a badge. Show me. If it's valid I show yellow and you pass. If not — what do I do?"

Students say "don't open it." Press: "Is doing nothing the right answer? What if someone is trying to break in?" Someone says "raise the alarm."

> **Script:** "Two outcomes, and I must decide what happens to **both**. Most beginners only plan the happy path."

Run four students through. For the invalid ones, do **nothing** the first time.

> **Script:** "Did anything happen? No. Was that a decision? … Half a decision. The machine didn't say yes. It also didn't say no."

GUIDED INSTRUCTION — `IF` VS `IF-ELSE` (10 MIN)

Step 1 (3 min). Build `if <(pressure) < 50> then [ say "WARNING" for 2 s ]`. Run at 80, then 40.

> **Script:** "At 80, nothing. At 40, a warning. The test failed and the program did **nothing at all**. Acceptable here?"

Step 2 (3 min). Open the `else`, add `say "NOMINAL"`.

> **Script:** "Read it as a sentence. *If pressure is under 50, warn. Else, say nominal.* Two outcomes. No gaps. This block is the difference between a system that reports and one that goes silent when something is wrong."

Name it: **the silent failure.** It returns in Term 3.

Step 3 (4 min). Comparison operators, as a table:

TEST	TRUE WHEN	WATCH OUT
`pressure < 50`	less than	not true at exactly 50

TEST	TRUE WHEN	WATCH OUT
`pressure > 50`	greater than	not true at exactly 50
`pressure = 50`	exactly	`=` **here is a question, not an instruction**
`pressure ≠ 50`	anything else	often the one you want

> **Script:** "Same symbol, two jobs. `set power to 50` **does**. `if power = 50` **asks**. Say it: *set means do, equals means ask.*"

INDEPENDENT PRACTICE — STUDIO TASK: AIRLOCK CONTROL (20 MIN)

Task A (5 min). Five-row test table, filled in before coding. Row 3 is exactly 50 — they must decide whether that is safe.

Task B (10 min). Build the airlock: below 50 → closed, alarm, `"DEPRESSURISATION"`; else → open, `"CLEAR"`.

Task C (5 min). Run all five cases. PASS/FAIL recorded. A failing test gets fixed, not explained.

Save as `P2_02_Else`.

DEBUGGING & DISCUSSION — "THE BRANCH THAT DOES NOTHING" (5 MIN)

Show the empty `else`.

> **Script:** "Nothing is broken. It behaves. But there's a hole — a state where the machine makes no statement. Find the hole. It's the empty branch."

Then: **the condition tests the wrong variable.** Copy-pasted from last week, never renamed.

> **Script:** "The code is perfect. The test is perfect. It's testing the wrong thing. This bug cannot be found by reading the block — only by asking what the block is meant to test. That's why you wrote the table first."

Exit ticket: "*What is the difference between `if` and `if-else`? One sentence.*"

4. Differentiations

Supporting struggling students

- Build only the `if` half; add `else` after it works.
- Pre-place the comparison block with the variable inside — dragging a reporter into a slot is a motor-skill barrier, not a thinking one.
- Print the table with two rows done.

Advanced extension challenges

- Three-band alert (below 50 / 50–79 / 80+). Cannot be done with one `if-else` — they must nest.
- Badge check with a countdown on failure.
- Silent-failure audit of their Term 1 project.
- Boundary argument: `< 50` or `≤ 50`? Defend it, then check the code matches.

WEEK 3

Compound Conditions — and the Table That Runs Later

1. Lesson Info Block

Topic	`and` / `or` / `not`; decision tables; **writing KanKaung's four-state table**
Duration	40 minutes
Keywords	compound · decision table · four states · truth table · priority
Mastery statement	*"A big decision is several small tests joined together — and four small decisions make a machine behave."*
Materials	Devices · decision table A3 (one per table) · **four-state table A3 (one per table)** · `P2_03_Launch` · exit tickets

> **This is the most important lesson in the term.** The four-state table students write today is the table they implement in Term 3, Week 4. Do not skip it, do not shorten it, and keep the originals.

2. Teacher Preparation Checklist

- [ ] Print two A3 sheets per table: the **decision table** (launch criteria) and the **four-state table** (blank).
- [ ] Build `P2_03_Launch` with three indicator sprites.
- [ ] Write the launch rule on the board.
- [ ] Prepare the De Morgan trap for the last 3 minutes.
- [ ] Have a folder labelled **TERM 3 — KEEP** for the completed four-state tables.
- [ ] **Print the flowchart symbol poster is still on the wall from Term 1 W2.** If it has come down, put it back before this lesson.

> **Do not skip Part C.** This is the busiest lesson in the term — decision table, four-state table, then the flowchart — and the flowchart is the one that gets dropped when you run long. It is also the artefact Term 3 Week 4 depends on, alongside the table.
>
> **If you are behind, cut Part A to three combinations.** Never cut Part C. A team that leaves without a flowchart enters Term 3 with a table they cannot picture.

3. Step-by-Step Instructional Guide

WARM-UP — "LAUNCH OR NO-GO" (5 MIN)

Four condition cards: **PRESSURE OK · WIND OK · CREW READY · DAYLIGHT.**

Round 1 all green → launch. Round 2 pressure red → no-go. Round 3: three green, daylight red.

The room splits. Let them argue for 60 seconds.

> **Script:** "Three of you said launch, three said wait, and neither is wrong — because we never agreed on the rule. A decision isn't a feeling. It's a **rule you write down before the situation arrives.**"

GUIDED INSTRUCTION — BUILD THE TABLE, THEN THE CODE (10 MIN)

Step 1 (4 min). Fill the decision table with the class.

> **Script:** "Look at the YES rows. What do they have in common? … Every test is green. That's `and`. `and` **means all of them.** It is the AND gate you drew in Week 1, with an action attached."

Step 2 (3 min). Translate one row.

> **Script:** "Every `and` makes the test **stricter**. More `and`s, fewer launches. `or` means at least one — a much looser test. `not` flips a single test."

Post on the wall: `and` **= stricter** · `or` **= looser** · `not` **= flips**

Step 3 — the turn (3 min). Put up the blank four-state table.

> **Script:** "Now something different. Every rule so far had one test. KanKaung has **two sensors**, and each can be on the line or off it. That's not one decision — that's four. Fill this in. You are writing the machine's brain nine weeks before it has a body."

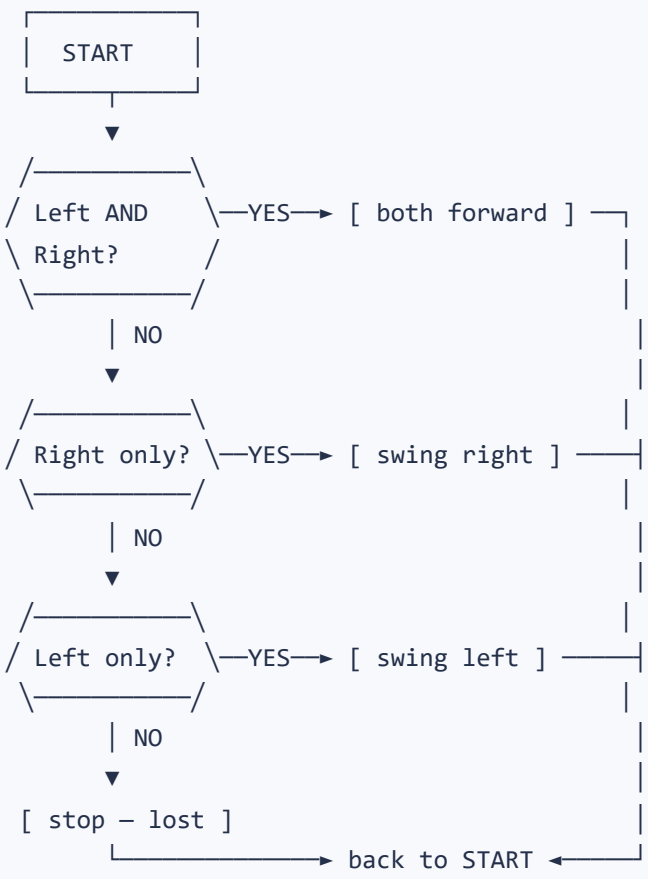
INDEPENDENT PRACTICE — STUDIO TASK: TWO TABLES (20 MIN)

Part A (8 min). Launch authority — decision table, then code. Test at least four combinations, including the one that split the warm-up.

Part B (7 min). The four-state table. Fill in all four rows by hand:

LEFT IR	RIGHT IR	MEANING	LEFT WHEEL	RIGHT WHEEL

Part C (3 min). Turn the table into a flowchart. One decision diamond per row, chained:



> **Script:** "Your table had four rows. This picture has three diamonds and one fall-through. Count the diamond exits: YES, YES, YES, and the last one has no NO — the bottom is 'none of the above'."

> **Script:** "This is why order matters. Ask 'right only' first and 'left only' second, and a robot that is centred never reaches the third diamond — it was caught by the first. **The picture shows you the order. The table can't.**"

Part D (2 min). One sentence under the table: *"If both sensors see nothing, the robot should ___ because ___."*

Save the app as `P2_03_Tables`. Hand the four-state table AND the flowchart to the teacher — both go in the TERM 3 folder.

Do not let them code the four-state table. Today it is paper only. The temptation to "just try it" spoils the Term 3 reveal and teaches nothing extra.

DEBUGGING & DISCUSSION — "TWO WAYS TO SAY THE SAME THING" (5 MIN)

Project `not <A and B>` versus `<not A> and <not B>`. Run both. Different results.

> **Script:** "These look like the same sentence. They are not. *Not (A and B)* means 'not both at once'. *Not A and not B* means 'neither one'. You will meet this again in Term 3, in a real machine, and it will cost you a run."

Then the four-state table:

> **Script:** "Look at your table. Row four says both sensors see nothing. Half of you wrote 'stop', half wrote 'keep going'. Both are defensible — and you'll find out which one is right when the robot meets a gap in the line."

Exit ticket: *"Give one example where `or` is right and `and` would be wrong."*

4. Differentiations

Supporting struggling students

- Two conditions only for the launch table.
- Give the four-state table with the "centred" row filled in; they complete three.
- Physical decision table: four cards laid out in a square, they place YES/NO cards.

Advanced extension challenges

- Priority conditions: "if pressure is critical, ignore everything else."
- Operator audit for each condition — `and`, `or`, or nesting, with a written reason.
- Build a condition that looks right and fails exactly one row. Swap with another team.

WEEK 4

Variables: The Labelled Box

1. Lesson Info Block

Topic	Creating, setting, changing, displaying; clamping
Duration	40 minutes
Keywords	variable · name · value · set · change · display · clamp · global
Mastery statement	*"A variable is a labelled box, and the label belongs to me."*
Materials	Devices · one box and 100 tokens per team · `P2_04_Power` · exit tickets

2. Teacher Preparation Checklist

- [ ] Prepare the physical prop: a **labelled box** ("POWER") and 100 tokens per team.
- [ ] Build `P2_04_Power`: four systems, four toggles, a large `power` display, a TICK button.
- [ ] Post the variable names: `power`, `minutes`, `mode`. Lower case, one word, meaningful.
- [ ] Prepare the broken demo: `power` reset to 100 inside the loop.

3. Step-by-Step Instructional Guide

WARM-UP — "KEEP THE TALLY" (5 MIN)

> **Script:** "I'll ring this bell some number of times while you recite the four systems. Tell me the total at the end. Ready?"

Ring 7 times while they recite. Most lose count.

> **Script:** "You dropped the number because your brain was doing two jobs. That's what a variable is for — a labelled box you can come back to."

Hold up the POWER box.

> **Script:** "Name, value, and it can change. Every score and battery bar in every game is one of these."

GUIDED INSTRUCTION — CREATE, SET, CHANGE, SHOW (10 MIN)

Step 1 (3 min). Create `power`. Point at the naming dialog.

> **Script:** "One word, all lower case, no spaces, meaningful. `power` is good. `p` is bad — in three weeks you'll have three p's."

Step 2 (2 min). `show variable [power]`.

> **Script:** "If you can't see it, you can't debug it. Show every variable you're working on, always."

Step 3 (3 min). `change [power] by [-4]` with a low-power check.

> **Script:** "`change` adds. `set` replaces. *Set* is a statement — 'power is 100'. *Change* is an event — 'power went down by 4'. Use the wrong one and the number never moves."

Step 4 (2 min). Click 30 times. `power` goes negative.

> **Script:** "The number is impossible. The machine doesn't know that. It has no idea what 'power' means — only arithmetic. **You** have to teach it the rule."

INDEPENDENT PRACTICE — STUDIO TASK: POWER GRID (20 MIN)

TASK	REQUIREMENT	TIME
A	`power` starts at 100, displayed	3 min
B	Four toggles, each spending its cost per tick	7 min
C	TICK advances a minute and deducts total cost	4 min
D	**Clamp:** never below 0; warn once at 0	4 min
E	`minutes` counts ticks and displays	2 min

Save as `P2_04_Power`.

Task D is the lesson: the first conditional written to protect a variable. Half the class will write it so the warning fires every tick forever. Ask: "How do you make it warn *once*?"

DEBUGGING & DISCUSSION — "THE NUMBER THAT NEVER MOVES" (5 MIN)

Show `set power to 100` inside the loop.

> **Script:** "Every tick it resets. It never falls. And it *looks* like it's working — the display shows 100 and the display is telling the truth. The program is lying about the budget."

Then local vs global:

> **Script:** "Two sprites, both with a variable called `power`. **They are not the same variable.** One goes down, the other doesn't. If you need one shared number, make it global."

Board: **same name ≠ same box.**

Exit ticket: "*Difference between* `set power to 0` *and* `change power by -1`*?*"

4. Differentiations

Supporting struggling students

- Tasks A–C only.
- Keep the physical box on the desk; remove a token for each spend before clicking.
- Pre-create `power` in `P2_04_Starter`.

Advanced extension challenges

- Clamp with a single warning that re-arms above 25 (hysteresis, arriving early).
- Auto-shed: lighting off below 30, comms below 15.
- Naming convention for the whole console, applied retroactively.

WEEK 5

Variables That Matter — The Trim Bench

1. Lesson Info Block

Topic	Parameters as live dials; calibration by measurement
Duration	40 minutes
Keywords	parameter · trim · calibration · offset · drift · tolerance · measurement
Mastery statement	*"I can change how a machine behaves without rewriting what it decides."*

Materials	Devices · **4 KanKaung robots (tables 1–4)** · **1 m taped corridor per table** · ruler · `P2_05_Trim` · exit tickets

> **Robot session 1 of 2.** Tables 1–4 today; tables 5–8 in Week 7.

2. Teacher Preparation Checklist

- [ ] Tape a **1 m corridor** on the floor for every table — two parallel lines, 40 cm apart.
- [ ] Pair and charge all four robots **before** the lesson.
- [ ] Run each robot straight. Two will curve left, three right, two will be nearly straight, one will be terrible. **Know which is which.**
- [ ] Prepare `P2_05_Trim`: `baseSpeed`, `trimL`, `trimR` at the top, plus a `drive` custom block.
- [ ] Print the calibration worksheet: five runs, drift measured, adjustment made.
- [ ] Cap speed at 55 for this lesson.

3. Step-by-Step Instructional Guide

WARM-UP — "WALK THE LINE" (5 MIN)

Two students walk side by side across the room, eyes forward, no talking. They drift apart.

> **Script:** "You both tried to walk straight and you ended apart. Why? Because you are different people. You didn't fail — you are **not identical**, and neither are these two servos."

> **Script:** "One is about 4% faster. Over a metre that's four centimetres of curve. Over five metres the robot is in the wall. Today you measure that 4% and subtract it."

GUIDED INSTRUCTION — ONE DIAL (10 MIN)

Step 1 (3 min). Replace four literals with variables.

`set [baseSpeed] to 55` → `drive (baseSpeed + trimL) (baseSpeed - trimR)`

> **Script:** "The logic is fixed. The **tuning** is separate. That separation is the whole point — you'll change these numbers fifty times today and never touch the program."

Step 2 (4 min). The corridor test. Run one metre, mark where it ends.

> **Script:** "It went forward. Did it go *straight*? Look at the start line and the end line. That's not a straight line — that's a curve with good manners."

Adjust `trimL` to `-8`. Run. Straighter.

> **Script:** "Eight. Not seven, not nine — you measured, you tested, you adjusted. That is **calibration**, and in Term 3 you will do it again on a sensor."

Step 3 (3 min). Clamping.

> **Script:** "The motor takes −100 to 100. Nothing stops you sending 150. Best case it ignores you; worst case it behaves strangely and you blame your code."

INDEPENDENT PRACTICE — STUDIO TASK: STRAIGHT AND TRUE (20 MIN)

RUN	TRIML	TRIMR	DRIFT AT 1 M	VERDICT
1	0	0		
2				
3				
4				
5				

Target: drift under 3 cm over one metre.

Rules: change **one** value per run; run the full metre every time; record the number before you run.

Then: **three consecutive runs** with the final values, all three recorded. One good run means nothing.

Save as `P2_05_Trim`.

DEBUGGING & DISCUSSION — "IT WAS STRAIGHT YESTERDAY" (5 MIN)

> **Script:** "Yesterday it drove straight. Today it curves. Nothing in the code changed. What did?"

Steer to: battery level, surface, a wheel pushed on crooked, a stripped gear.

> **Script:** "Trim is not a permanent number. It's a **snapshot**. Write your values in the notebook and check them at the start of every lesson. Real robots are re-trimmed every session — that's maintenance, not a fault."

Exit ticket: "*Your robot curves right. Which trim value changes, and which way?*"

4. Differentiations

Supporting struggling students

- Give them the trim value you measured; they record and verify.
- Shorten the corridor to 50 cm.
- Accept "much better" plus three runs if measurement is unreliable.

Advanced extension challenges

- Trim at three speeds and explain why the value differs.
- Turn calibration: find the values that make a 90° turn actually 90°.
- Write the maintenance card — three lines any student can follow to re-trim.

Robot-table adjustments

- One driver at a time, seated, on the taped corridor only.
- If a robot will not pair in 60 seconds, swap to the spare.
- Charged batteries matter more this week than any other — weak cells change servo timing and ruin the data.

WEEK 6

Modes: One Variable Runs Everything

1. Lesson Info Block

Topic	A mode variable driving all behaviour; legal transitions
Duration	40 minutes
Keywords	state · mode · transition · diagram · ownership · illegal move
Mastery statement	"*One variable can tell the whole system what to do.*"

Materials	Devices · four A4 mode signs (IDLE / ACTIVE / CHARGING / EMERGENCY) in the corners · mode diagram worksheet · `P2_06_Modes` · exit tickets

2. Teacher Preparation Checklist

- [ ] Tape the four signs in the four corners before students enter.
- [ ] Build `P2_06_Modes`: base sprite, four costumes, `mode` variable, four buttons.
- [ ] Prepare the broken demo: **two scripts writing to** `mode`, causing flicker.
- [ ] Write on the board: **mode decides what happens. Events change the mode. Nothing else.**

3. Step-by-Step Instructional Guide

WARM-UP — "FOUR CORNERS" (5 MIN)

> **Script:** "IDLE there. ACTIVE there. CHARGING there. EMERGENCY there. When I call a mode, walk to it. When I call a transition, move once — only if the rule allows."

Add rules: **"ACTIVE → CHARGING → EMERGENCY → ACTIVE"** — the last transition is illegal. Students move anyway.

> **Script:** "Three of you walked from EMERGENCY to ACTIVE. That transition doesn't exist. In a real machine, that's how a stop gets cancelled by accident. **The rules about moving between states matter as much as the states.**"

GUIDED INSTRUCTION — DIAGRAM FIRST (10 MIN)

Step 1 (4 min). Four circles on the board; fill the arrows with the class, each labelled with its trigger.

> **Script:** "This picture is the design. Notice we already found a bug — there's no arrow out of EMERGENCY except this one."

Step 2 (3 min). The mode variable and a loop of `if` blocks.

> **Script:** "Four checks, one loop, one variable. The loop asks 'what mode am I in?' sixty times a second and does the right thing. With events alone you'd need a script for every pair of transitions — that explodes."

Step 3 (3 min). Ownership.

> **Script:** "Who owns the mode? If two scripts write to it, the last one wins, and which one is last is not something you control. **One owner per variable.** Write it in the diagram."

INDEPENDENT PRACTICE — STUDIO TASK: BASE STATE MACHINE · CARD 7 (20 MIN)

TASK	REQUIREMENT	TIME
A	Mode diagram complete, every arrow labelled — teacher initials it	5 min
B	IDLE, ACTIVE, CHARGING implemented	10 min
C	EMERGENCY triggered by `power < 20`, overrides everything, only RESET escapes	5 min

Save as `P2_06_Modes`.

The hard part of C: EMERGENCY is a *transition*, not a mode. Students put the `power < 20` check in a button script and it never fires, because nothing presses a button when power drops.

> **Script:** "Where does the check live? In the same loop that reads the mode. That loop runs all the time. The button doesn't."

DEBUGGING & DISCUSSION — "TWO THINGS CHANGING THE MODE" (5 MIN)

Show the flicker demo.

> **Script:** "Who owns it? Two writers, and you don't control which one wins."

Second: **the mode never changes** because the transition rule was never wired.

Exit ticket: "*Why is one mode variable better than four separate on/off switches?*"

4. Differentiations

Supporting struggling students

- Two modes: IDLE and ACTIVE.
- `P2_06_Starter` with the loop and `if` blocks placed, actions empty.
- Walk them through the diagram physically before coding.

Advanced extension challenges

- Five states with SLEEP.
- Illegal-transition guard that refuses and logs.
- Automatic transitions after a timeout.

WEEK 7

Functions — and the Countdown

1. Lesson Info Block

Topic	Custom blocks; naming and reusing a routine; **first parameterised routine on the robot**
Duration	40 minutes
Keywords	function · custom block · define · call · reuse · abstraction · parameter
Mastery statement	*"I write it once and call it anywhere."*
Materials	Devices · **4 KanKaung robots (tables 5–8)** · `P2_07_Duplicate` · sticky notes · exit tickets

> **Robot session 2 of 2.** Tables 5–8 today. Tables 1–4 do the countdown as a sprite-only project and watch.

2. Teacher Preparation Checklist

- [ ] Build `P2_07_Duplicate`: the same 12-block routine written out four times.
- [ ] Practise creating a custom block with a parameter — the dialog is where students stick.
- [ ] Confirm whether custom blocks can be called from other sprites.
- [ ] Print the signature card format.

3. Step-by-Step Instructional Guide

WARM-UP — "THE INSTRUCTION READER" (5 MIN)

Four students each read the same six-step card aloud.

> **Script:** "Four cards, same six steps, four times. Now — one card with a name on it. I say the name, you do the routine."

> **Script:** "Four words instead of twenty-four. And when I change one line on the card, all four uses update at once. With four cards I'd change four things and forget one."

GUIDED INSTRUCTION — MAKE A BLOCK (10 MIN)

Step 1 (3 min). Show the duplication. Count the blocks; count the ideas.

> **Script:** "Forty-eight blocks. One idea. That's **duplication** — the same thought copied instead of named."

Step 2 (4 min). Select, "Make a Block", name it `reportStatus`.

> **Script:** "A good name says what it does, not what it contains. If you can't name it in one word, it's probably doing two things."

> **Script:** "This is **abstraction**. The name hides the forty blocks inside, so you can think about *when to call it* instead of *what is in it*. Every block you already use — `move`, `say`, `wait` — is exactly this: a name standing in front of code you did not write."

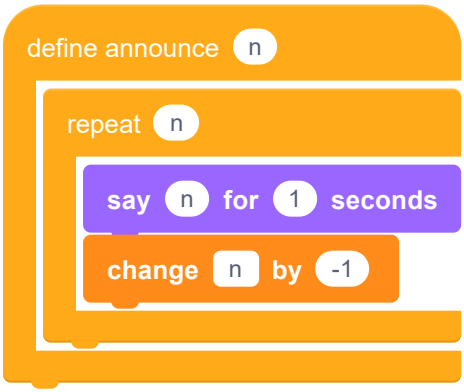
Step 3 (3 min). Add a parameter: `announce (message, seconds)`.

> **Script:** "Now it's not about one message — it's about **any** message. Same routine, many cases."

INDEPENDENT PRACTICE — STUDIO TASK: LIBRARY AND COUNTDOWN (20 MIN)

All tables (10 min): refactor `P2_07_Duplicate` into custom blocks. Write a signature card for each.

Robot tables (10 min): build `announce (n)` — say the number, wait a second, repeat down to zero — then drive forward for one second.



> **Script:** "Same block, different starting number. Call it with 3, with 5, with 1. One routine, three behaviours."

Save as `P2_07_Library`.

DEBUGGING & DISCUSSION — "IT'S PERFECT AND NOTHING HAPPENS" (5 MIN)

Show a beautiful custom block that is never called.

> **Script:** "Defining a block is writing the recipe. Calling it is cooking. You can have a perfect recipe and still be hungry."

Exit ticket: "*Your program has 40 blocks and 1 idea. What should you do?*"

4. Differentiations

Supporting struggling students

- Refactor two copies, not four.
- Give the exact block name.
- On the robot, use a fixed 3-second countdown — no parameter.

Advanced extension challenges

- Library audit of every custom block.
- Split one block that does two unrelated things.
- Interface-first: write the signature, then build to match it.

WEEK 8

Parameters: One Block, Many Cases

1. Lesson Info Block

Topic	Passing values into a routine; generalising vs hard-coding
Duration	40 minutes
Keywords	parameter · argument · generalise · hard-coded · interface · reuse
Mastery statement	"*One block, many cases.*"
Materials	Devices · `P2_08_Servo` (four hard-coded routines) · interface contract cards · exit tickets

2. Teacher Preparation Checklist

- [ ] Build `P2_08_Servo` with four custom blocks: `scan_left`, `scan_centre`, `scan_right`, `scan_hard_left` — each hard-coded to a different angle. They work. That is the point.

- [ ] Confirm parameter typing in Padma — numbers only, or text too?
- [ ] Prepare the broken demo: a routine with a parameter it never uses.

3. Step-by-Step Instructional Guide

WARM-UP — "THE VENDING MACHINE" (5 MIN)

> **Script:** "This machine gives you something. Press A." Mime handing over a fixed item. "Now I want a *different* thing. What do I need? … Not a new machine. A **choice**."

Point at the slot.

> **Script:** "That slot is the input. Same machine, different result. Today you build one block that works in ten places instead of ten blocks that work in one."

GUIDED INSTRUCTION — FOUR BLOCKS TO ONE (10 MIN)

Step 1 (3 min). Show the four hard-coded blocks. The only difference is one number.

Step 2 (4 min). Open one, add a parameter `angle`, delete the hard-coded value, drag the parameter into its place.

> **Script:** "The parameter is a hole. Before, the number was welded in. Now it's a socket."

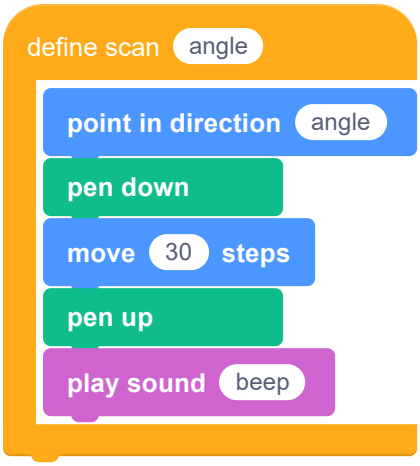
Step 3 (3 min). Call it with 30, 90, 150.

> **Script:** "One block. Three behaviours. And a fourth costs nothing — you just call it with a different number."

INDEPENDENT PRACTICE — STUDIO TASK: THE RADAR SWEEP (20 MIN)

Instead of refactoring four routines one at a time, build **one block that sweeps a whole arc** — and watch it draw.

TASK	REQUIREMENT	TIME
A	Build `scan (angle)`: turn to the angle, wait, take the reading, draw a line from centre to that point	8 min
B	Call it in a loop from 0 to 180 in steps of 15 — **a radar sweep**	6 min
C	Write the signature in the manual, including the 0–180 range	3 min
D	Press a key to clear the stage and sweep again	3 min



> **Script:** "Four hard-coded blocks became one. Now put it in a loop and it becomes something none of those four could do — **a sweep**. One block, thirteen angles, and you can see every one of them."

Task B produces the moment of the lesson. When the arc appears on the stage, the class will understand why parameters exist — not because you explained it, but because the picture required it.

Signature format: `scan (angle)` — points to `angle`, draws to it, and records. Range 0–180. Outside that, the block does nothing and reports an error.

Save as `P2_08_Params`.

DEBUGGING & DISCUSSION — "THE SOCKET NOBODY USED" (5 MIN)

Show a routine that takes a parameter and ignores it.

> **Script:** "The parameter exists, it's connected, and it's completely ignored. Adding a parameter does not automatically use it."

Second: **calling with an impossible value.** `scan 900` runs, does nothing, reports success.

> **Script:** "That's a **range** bug. The machine has limits and your code doesn't know them. Write the range in the signature, and check it inside the block."

Look at the sweep on the stage.

> **Script:** "Who has a gap in their arc? … You called `scan` with 15, 30, 45 — but you forgot 0 or 180. **A sweep shows you the parameters you skipped.** A table wouldn't."

Exit ticket: "`draw_square`, `draw_triangle`, `draw_hexagon` — *what single parameterised block replaces all three?*"

4. Differentiations

Supporting struggling students

- Two routines instead of four.
- Give the parameter pre-created, empty.
- Numbers only; skip the text parameter.

Advanced extension challenges

- Multi-parameter routine: `log (value, label, priority)`.
- Defaults by hand: passing `0` means "wait for a tap".
- Parameter validation with an error report.
- Interface-first design: write the signature, then build to match.

WEEK 9

Integration: Two Teams, One Console

1. Lesson Info Block

Topic	Merging independently built subsystems with agreed messages
Duration	40 minutes
Keywords	integration · interface · contract · message bus · ownership · conflict
Mastery statement	*"Agree the boundary before you build on both sides of it."*
Materials	Devices · interface contract sheets · `P2_09_Broken` · exit tickets

2. Teacher Preparation Checklist

- [ ] Decide the pairs before class and write them on the board.
- [ ] Build `P2_09_Broken`: a mismatched message name plus two scripts resetting `power`.
- [ ] Print contract sheets with four boxes: **WHO SENDS · WHAT · WHAT'S IN IT · WHO LISTENS.**
- [ ] Seat paired teams facing each other, not side by side.

3. Step-by-Step Instructional Guide

WARM-UP — "TWO TEAMS, ONE DOOR" (5 MIN)

Split the room with tape.

> **Script:** "Left, you control the door. Right, you control the alarm. You may not speak to each other. Sixty seconds: the alarm must ring when the door opens."

Chaos.

> **Script:** "Why did that fail? You couldn't agree what 'door opened' means. **Integration is a conversation, not a merge.**"

GUIDED INSTRUCTION — THE CONTRACT (10 MIN)

Fill the sheet on the board for the door example. Then post the four rules:

1. Every message has an owner.
2. Every variable has an owner.
3. Only the owner writes. Everyone else reads.
4. Spelling is the contract.

INDEPENDENT PRACTICE — STUDIO TASK: MERGE THE CONSOLE (20 MIN)

Task A (6 min). Both teams complete and sign one contract. Teacher initials.

Task B (10 min). Merge. No two scripts write the same variable.

Task C (4 min). Three scenarios: normal, low power, and a message nobody listens for — **must not crash**.

Save as `P2_09_Console`.

DEBUGGING & DISCUSSION — "WHO BROKE IT?" (5 MIN)

> **Script:** "Team A says B broke it, B says A broke it. How do we settle this without arguing?"

Walk the arbitration: is it sent, is the spelling identical, is anyone else listening, who writes the shared variable, does it work with half removed.

> **Script:** "What resolved it wasn't opinion — it was the contract and a bisection."

Exit ticket: "*Difference between a message and a variable?*"

4. Differentiations

Supporting struggling students

- One message and one variable.
- Pre-fill the contract; they implement the receiver.

Advanced extension challenges

- Three-way merge.
- Version the contract and show what renaming breaks.
- Fault tolerance: keep running if one subsystem stops responding.

WEEK 10

Testing: Normal, Boundary, Hostile

1. Lesson Info Block

Topic	Test design; boundary values; deliberate sabotage
Duration	40 minutes
Keywords	test case · expected · boundary · edge case · hostile · regression
Mastery statement	*"I test the values that will break it, not the ones that flatter it."*
Materials	Devices · test table template · sabotage cards · Term 1 Bug Wall poster · exit tickets

2. Teacher Preparation Checklist

- [ ] Print the test table with **NORMAL · BOUNDARY · HOSTILE** bands.
- [ ] Print 8 sabotage cards, one fault each:
  - set `power` to 0 · set `power` to 101 · rename a message listener · press RESTOCK when power is already 100 · hold EMERGENCY and press everything · hold TICK until power reaches 0, then tick again · set a variable to text · leave it running 3 minutes
- [ ] Decide the sabotage pairing — teams must sabotage a team they have not worked with.

3. Step-by-Step Instructional Guide

WARM-UP — "BREAK MY PROGRAM" (5 MIN)

Project a working program.

> **Script:** "Two minutes to make it fail. You may not touch the code — only use it strangely."

They press fast, press nothing, press during an alarm.

> **Script:** "Every failure you found, you found by doing something the programmer didn't imagine. That's **hostile testing**, and your users will do exactly what you just did — without trying to help."

GUIDED INSTRUCTION — THE THREE BANDS (10 MIN)

Normal (2 min). The obvious case. "The test that was already passing."

Boundary (4 min).

> **Script:** "Your rule says *below 20, warn*. What happens at exactly 20? At 19? At 21? Three tests, one rule. Almost every real bug lives in that narrow band."

Board: **for every `<` you wrote, test the value, the value − 1, and the value + 1.**

Hostile (4 min). Power of −5. Power of 500. Power as the word "banana". RESTOCK at 100.

INDEPENDENT PRACTICE — STUDIO TASK: THE TEST BENCH · CARD 8 (20 MIN)

Task A (7 min). Eight rows minimum, three bands. **Task B (5 min).** Run your own suite. Fix what fails. **Task C (8 min).** Swap devices, draw a sabotage card, inject the fault, write a **bug report**.

> *"We set `power` to 0. Expected: CRITICAL. Actual: console froze. Severity: high."*

Save as `P2_10_Test`. Manual paragraph: "*Known limitations: …*"

DEBUGGING & DISCUSSION — "THE BUG THAT ONLY APPEARS AT ZERO" (5 MIN)

Project the most interesting failure. Praise it publicly.

> **Script:** "This bug existed for six weeks. Every normal test passed. It only appears at zero. **This is why testing isn't the last step — it's the step that tells you what you actually built.**"

Then the term close:

> **Script:** "Ten weeks ago you couldn't make a thing decide. Today you have resources, modes, reusable routines, contracts between subsystems, and a test suite that tells you the truth. Next term you put all of it on four wheels."

Exit ticket: "*Write one boundary test for a rule using `< 50`.*"

4. Differentiations

Supporting struggling students

- Six rows; boundary band only.
- Pre-match the sabotage card to their partner's known weakness.
- Pre-write the bug report format with blanks.

Advanced extension challenges

- Regression suite for Term 3's changes.
- Adversarial pairing: design a card for a team you have not seen.
- Failure-mode table: does each subsystem fail loudly or silently?

Term 2 at a glance

WK	LESSON	NEW CONCEPT	BUILD
1	The Decision Problem	Conditions, measurable facts	Edge-aware patrol
2	If / Else	Branches, comparisons	Airlock
3	Compound Conditions	`and` / `or` / `not` + **four-state table**	Launch + **KanKaung's brain on paper**
4	Variables	Set, change, clamp	Power grid
5	Variables That Matter	Trim, calibration	**Robot drives straight** *(tables 1–4)*
6	Modes	State machine	Four-mode console
7	Functions	Custom blocks	**Countdown on the robot** *(tables 5–8)*
8	Parameters	Generalising routines	Shared scanner
9	Integration	Contracts	Merged console
10	Testing	Boundary values	Test suite

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

1. Say which statements a machine can and cannot test.
2. Write a condition and say what happens when it fails.
3. Explain `set` versus `=`.
4. Join tests with `and` / `or` / `not` and predict the outcome.

5. **Fill in the four-state table from memory.**
6. Create, update, display and clamp a variable.
7. Drive a whole system from one mode variable.
8. Name a repeated routine and call it in more than one place.
9. Pass a value into a routine so one block serves many cases.
10. Write a test table that includes the boundary value.

Before Term 3 begins

- [ ] Every project saved and copied to the class SD card.
- [ ] **Four-state tables collected and filed in the TERM 3 folder.** This is the Term 3 Week 4 lesson; losing them costs a week.
- [ ] **Four-state flowcharts collected and filed with the tables.** Term 3 Week 4 uses both — the table tells the team *what* to do, the flowchart tells them *in what order*. A team with only the table will build four `if` blocks and get the order wrong.
- [ ] Notebooks collected, one page initialled per week.
- [ ] Trim values recorded per robot in the teacher log.
- [ ] Robots charged, wheels checked, servos tested.
- [ ] Exit tickets sorted; the "confused" pile drives Term 3 Week 1.

> **Count them before you close the folder:** one table and one flowchart per team. If a team has only one, that lesson failed somewhere in Week 3 — re-run Part C at the start of Term 3 Week 1 rather than discovering it in Week 4.

Term 2 pack v2.0 · Padma Robotics Programme · rewritten to match the curriculum map (RULES term)