BRIDGE TO TEXT — LESSON PLANS

12 weeks · Two sessions per week · Ages 13–15

Format: 2 × 40 minutes per week · **Devices:** one computer per student **Prerequisite:** Padma Terms 1–3 · **Exit:** ready to begin iGCSE Computer Science

> **How this pack differs from the Padma packs.** Each week is **two sessions**, because notation does not fit in forty minutes. Session A proves equivalence — the Python version does what the blocks did. Session B changes the requirement so the text version does something the blocks could not.
>
> The four-part template becomes five sections: Info, Preparation, Session A, Session B, Differentiation.

Before Week 1

If you have never written Python, read this.

This course assumes the teacher can read a traceback and knows why `"5" + 5` fails. If that is not yet true, the honest options are in the framework (§11): 40–60 hours of personal Python before term, or co-teaching with someone who has it.

What you need on every machine, before Week 1:

- Python 3.8 or later, installed offline
- An editor with line numbers (IDLE, Thonny, or VS Code — anything)
- A folder per student, named `SURNAME_Firstname`
- The **transfer table** printed and on the wall (§4 of the framework)

One rule that saves the term: students type code, they do not copy-paste. Typing is where syntax is learned.

The wall reference

PADMA	PYTHON
`when ⚑ clicked`	the first line of the file

PADMA	PYTHON
`repeat 4`	`for i in range(4):`
`forever`	`while True:`
`repeat until <c>`	`while not c:`
`if / else`	`if … : / elif … : / else … :`
`set x to 5`	`x = 5`
`change x by 1`	`x += 1`
`and / or / not`	`and / or / not`
`define f (a)`	`def f(a):`
`say [t]`	`print("t")`
`ask and wait`	`input()`

WEEK 1

Blocks Become Text

1. Lesson Info Block

Topic	The editor, `print()`, running a file, indentation
Duration	2 × 40 minutes
Keywords	statement · print · string · indentation · run · output
Mastery statement	*"I can make a computer say something, in text."*
Materials	One computer per student · Python + editor installed · transfer table on the wall · printed "first program" card

2. Teacher Preparation Checklist

- [ ] Test the Python installation on the **oldest** machine. Run a one-line file.

- [ ] Confirm every student can create, save and run a `.py` file before the lesson ends. This is the whole week.
- [ ] Print the transfer table at A3.
- [ ] Have one deliberately broken file ready: a `print` with a missing bracket.
- [ ] Write on the board: **indentation is meaning.**

3. Session A — Port (40 min)

Show the blocks (5 min). Project the Week 1 Padma script — a hat block and `say "Hello"`.

> **Script:** "In Padma that took two blocks and a green flag. Here it takes one line, and there is no flag. **The file runs when you run it.** That is the first difference and the biggest one."

Live port (10 min). Type, do not paste:

```
print("Hello, KanKaung")
```

Run it.

> **Script:** "Read it left to right. `print` is the verb. The brackets hold what to print. The quotes mean *this is text, not a command*. Remove a quote and see what happens."

Delete one quote, run, show the `SyntaxError`. Undo.

Student port (20 min). From the transfer table, closed book on Python:

TASK	REQUIREMENT
A	Print your own name
B	Print three lines: name, term, one thing you built
C	Print a line containing quotation marks inside the text
D	Print your name on one line and your school on the next, with no blank lines

Compare (5 min). One student's screen on the projector.

> **Script:** "Two people wrote this four different ways. All correct. Python does not care how it looks to *you* — it cares about the brackets and the quotes."

4. Session B — Extend (40 min)

Recap (5 min). "What did the error message say when I deleted the quote?"

One new element (10 min). Indentation. Show. In the first block, `·` marks a space and is not typed:

```
print("start")
····print("indented")   ← 4 spaces
print("end")
```

> **Script:** "This is a `SyntaxError` and it looks harmless. In Padma, indentation was cosmetic. Here it is **structure**. Whitespace tells Python what belongs to what."

Show the working version — no leading spaces on any line:

```
print("start")
print("indented")
print("end")
```

Change the requirement (20 min). They cannot do loops yet, so the extension is *output control*:

- Print a framed banner of your name, using `*` characters, three lines
- Print a blank line between two paragraphs (`print()`)
- Print a line of exactly 40 `=` characters

Exit ticket (5 min). "*What are two things that can go wrong on your very first line of Python?*"

5. Differentiations

Supporting struggling students

- Pre-create the file with the first line typed; they add two more.
- Sit with them for the first 60 seconds of the port — the editor, not the code, is the barrier.
- Accept three working lines instead of four tasks.

Advanced extension

- Print a 3-line ASCII box using only `print`.
- Explain in one sentence why `print("5" + "5")` gives `55` and `print(5 + 5)` gives `10` — do not fix it, just notice it.
- Write the shortest program that prints a blank line.

WEEK 2

Variables and Input

1. Lesson Info Block

Topic	Assignment, `input()`, `int()`, f-strings, types
Duration	2 × 40 minutes
Keywords	variable · assignment · input · type · int · string · f-string
Mastery statement	*"I can store a value, change it, and ask the user for one."*
Materials	Computers · the power-grid script from Padma Term 2 W4 on the projector

2. Teacher Preparation Checklist

- [ ] Project the Padma power-grid script side by side with its Python version.
- [ ] Prepare the type-error demo: `"5" + 5`.
- [ ] Know the three errors you will see: `NameError`, `TypeError`, `ValueError`.
- [ ] Write on the board: `input()` **always gives you text.**

3. Session A — Port (40 min)

Show the blocks (5 min). The Padma version: `set power to 100`, `change power by -4`, `show variable`.

Live port (10 min).

```
power = 100
power = power - 4
print(power)
```

> **Script:** "`set power to 100` became `power = 100`. Read the second line — it is not maths, it is *'work out the right side, then put the answer in the box on the left'*. That is why the same name appears twice."

Add `power += 10` beside it and show it is the same as `power = power + 10`.

Student port (20 min). Port the power grid, output only:

TASK	REQUIREMENT
A	Start at 100, subtract 4, print the result
B	Subtract four times in a row; print after each
C	Add a `minutes` variable that counts how many times you subtracted
D	Print both values on one line

Compare (5 min). Show two solutions to D — `print(power, minutes)` and `print(f"{power} units, {minutes} min")`.

> **Script:** "The `f` before the quote means *fill in the blanks*. Everything in curly braces is code. Everything else is text."

4. Session B — Extend (40 min)

Recap (5 min). "Why does `power = power - 4` not mean power equals power minus four?"

One new element (10 min). `input()` and types.

```
name = input("Your name? ")
print("Hello, " + name)
```

Then the failure:

```
speed = input("Speed? ")
print(speed + 10)
```

> **Script:** "Input is always text. Even if they type 50, you get the characters `5` and `0`. Python will not add a number to text — and it tells you so."

Fix it:

```
speed = int(input("Speed? "))
print(speed + 10)
```

Change the requirement (20 min). Make the power grid interactive:

- Ask for a starting power, convert it with `int()`
- Ask how many minutes to simulate

- Print the final power and a warning if it is below 20 (they do not have `if` yet — print `"LOW"` or `"OK"` using a trick: `print("LOW" if power < 20 else "OK")` is allowed as a preview)

Exit ticket (5 min). *"What does `input()` always give you, and what do you do about it?"*

5. Differentiations

Supporting struggling students

- Give the f-string pre-written; they fill in the variable names.
- Skip Task D; two separate prints are fine.

Advanced extension

- Ask for two numbers and print their sum, on one line.
- Print a value to two decimal places (`:.2f`).
- Explain the difference between `5` , `5.0` and `"5"` in one sentence each.

WEEK 3

Selection

1. Lesson Info Block

Topic	`if` / `elif` / `else`, comparison, `and` / `or` / `not`
Duration	2 × 40 minutes
Keywords	selection · condition · branch · elif · comparison · boolean
Mastery statement	*"I can make a program take a different path."*
Materials	Computers · Padma airlock script · the four-state table from Term 2 W3

2. Teacher Preparation Checklist

- [ ] Project the Padma airlock (`if pressure < 50 then … else …`).
- [ ] Have the four-state table ready — it becomes an `if` / `elif` chain this week.
- [ ] Prepare the `=` vs `==` demo. This is the week's most common error.
- [ ] Write on the board: `=` **assigns.** `==` **asks.**

3. Session A — Port (40 min)

Show the blocks (5 min). The airlock: a C-shaped `if`, an `else` below it.

Live port (10 min).

```
pressure = 40
if pressure < 50:
    print("WARNING")
else:
    print("NOMINAL")
```

> **Script:** "Three things to notice. The colon at the end of the `if`. The four spaces on the next line. And there is no `end` — the indentation *is* the end. Delete the spaces and the program breaks."

Student port (20 min).

TASK	REQUIREMENT
A	Port the airlock exactly
B	Add a second test: below 20 is `"CRITICAL"` (needs `elif`)
C	Ask for the pressure with `input()` and `int()`
D	Test at 80, 51, 50, 49, 20, 0 and record what happens at exactly 50

Compare (5 min). Task D is the point.

> **Script:** "Who tested 50? What did you get? **The boundary is where bugs live.** In Padma you discovered this with a sprite; here you discover it with a number."

4. Session B — Extend (40 min)

Recap (5 min). "What does the colon do? What happens if you delete the indentation?"

One new element (10 min). `==` versus `=`.

```
mode = "PATROL"        # assign
if mode == "PATROL":   # ask
    print("patrolling")
```

Show the failure: `if mode = "PATROL":` → `SyntaxError`.

> **Script:** "Same symbol in Padma, two meanings. In Python they are different symbols, and the machine will not let you confuse them. That is a gift."

Change the requirement (20 min). Port the four-state table. This is the week's real work.

```
left = True
right = False

if left and right:
    print("centred: both forward")
elif right and not left:
    print("drifted right: slow left")
elif left and not right:
    print("drifted left: slow right")
else:
    print("lost: stop")
```

Then they test all four combinations by editing the two booleans.

> **Script:** "This is the table you wrote in Term 2, week 3. Four rows. Four branches. It is the same program you ran on KanKaung — just without wheels."

Exit ticket (5 min). *"Why `elif` rather than four separate `if`s?"*

5. Differentiations

Supporting struggling students

- Two branches only (open/closed), then add `elif` with help.
- Give the four-state `if` chain with the first branch complete.

Advanced extension

- Compound conditions with `not (a and b)` versus `not a and not b` — show they differ.
- Add a fifth state for "on a crossing" (both sensors see a wider line).

WEEK 4

Iteration

1. Lesson Info Block

Topic	`for`, `range()`, `while`, nesting
Duration	2 × 40 minutes
Keywords	iteration · for · while · range · nesting · loop variable
Mastery statement	*"I can repeat something a set number of times, or until something changes."*
Materials	Computers · the loop ladder from Term 1 W6

2. Teacher Preparation Checklist

- [ ] Have the Padma square (`repeat 4 [move, turn]`) on the projector.
- [ ] Prepare the infinite-loop demo — and know how to stop it (`Ctrl+C`).
- [ ] Print a `range()` reference strip: `range(4)`, `range(1,5)`, `range(0,10,2)`.

3. Session A — Port (40 min)

Show the blocks (5 min). The square, with `repeat 4`.

Live port (10 min).

```
for i in range(4):
    print("side", i + 1)
```

> **Script:** "`repeat 4` became `for i in range(4)`. `i` is a counter Python makes for you — it starts at 0 and goes to 3. That is why `i + 1` prints 1 to 4."

Show `range` variants.

Student port (20 min).

TASK	REQUIREMENT
A	Print "side 1" to "side 4" with a loop
B	Print the numbers 1 to 10
C	Print the even numbers from 0 to 20

TASK	REQUIREMENT
D	Print a 3×3 grid of `#` using a loop inside a loop

Compare (5 min). Task D.

> **Script:** "How many `print` statements would this need without loops? Nine. With loops? Two. That is the same lesson you learned in Term 1 with the lotus wall."

4. Session B — Extend (40 min)

Recap (5 min). "What does `range(4)` actually produce?"

One new element (10 min). `while` — the conditional loop.

```
power = 100
while power > 20:
    power -= 15
    print(power)
print("done")
```

> **Script:** "`for` is 'do this N times'. `while` is 'keep going while this is true'. In Padma these were `repeat N` and `repeat until` — same two ideas."

The infinite loop demo (3 min). Write `while True: pass`, run it, let it hang, then `Ctrl+C`.

> **Script:** "This is the one mistake that locks the machine. `Ctrl+C` in the console stops it. Every programmer has done this; most have done it today."

Change the requirement (20 min).

- Simulate the power grid: start at 100, subtract 4 per minute, stop when below 20, print how many minutes that took
- Then do it with `for` and compare which was clearer
- Then nest: for each of three systems, drain to zero and report

Exit ticket (5 min). "*Give one situation where `while` is right and `for` would be wrong.*"

5. Differentiations

Supporting struggling students

- `for` only this week; introduce `while` in Week 5.

- Give the grid task as two separate loops rather than nested.

Advanced extension

- FizzBuzz 1–20.
- A countdown with `while`, printing one line per second using `time.sleep(1)`.
- Explain why `while` can fail where `for` cannot.

WEEK 5

Functions

1. Lesson Info Block

Topic	`def`, parameters, `return`
Duration	2 × 40 minutes
Keywords	function · define · call · parameter · return · scope
Mastery statement	*"I write it once, name it, and call it anywhere."*
Materials	Computers · the routine library from Padma Term 2 W7–8

2. Teacher Preparation Checklist

- [ ] Project the Padma cloned-routine workshop (48 blocks, one idea).
- [ ] Prepare the "defined but never called" demo — the same bug as Term 2.
- [ ] Know the difference you will need to explain: `print` shows a value, `return` hands it back.

3. Session A — Port (40 min)

Show the blocks (5 min). The four duplicated scan routines.

Live port (10 min).

```
def scan(angle):
    print(f"scanning {angle} degrees")

scan(30)
scan(90)
scan(150)
```

> **Script:** "`define scan (angle)` became `def scan(angle):`. The parameter is inside the brackets. Calling it is the name with a value — and note the value can change each time."

Student port (20 min).

TASK	REQUIREMENT
A	Define `scan(angle)` and call it three ways
B	Define `announce(message)` and call it twice
C	Define `report(name, value)` and call it three times
D	Define a function, then **forget to call it** — run and observe

Task D on purpose.

Compare (5 min).

> **Script:** "Whose program printed nothing? Why? **Defining is not calling.** You wrote a recipe and never cooked. Same bug you met in Term 2 — and it is the most common function bug in every language there is."

4. Session B — Extend (40 min)

Recap (5 min). "What is the difference between a definition and a call?"

One new element (10 min). `return`.

```
def add_power(power, amount):
    new_power = power + amount
    return new_power

result = add_power(80, 15)
print(result)
```

> **Script:** "`print` shows a value to a human. `return` hands it back to the program. A function that only prints cannot be used in a calculation — and that is the difference between a demonstration and a tool."

Show the trap:

```
def add_power(power, amount):
    power + amount        # no return

result = add_power(80, 15)
print(result)             # None
```

Change the requirement (20 min).

- Write `clamp(value, low, high)` that returns the value limited to a range
- Write `low_power(power)` that returns `True` or `False`
- Use both in a small program that reads three power values and reports which are low

Exit ticket (5 min). *"What does a function return if you forget the `return`?"*

5. Differentiations

Supporting struggling students

- `print`-only functions; introduce `return` in Week 6.
- Give the `clamp` body and let them write the calls.

Advanced extension

- A function that returns two values.
- Explain scope: why a variable inside a function is invisible outside.
- Rewrite Week 4's nested grid as a function taking a size.

WEEK 6

Lists

1. Lesson Info Block

Topic	Creating a list, index, `append`, `len`, looping
Duration	2 × 40 minutes
Keywords	list · index · element · append · length · zero-index
Mastery statement	*"I can hold many values under one name."*
Materials	Computers · a physical row of 5 boxes or cards per table

2. Teacher Preparation Checklist

- [ ] **You are not introducing lists. You are introducing what Python does differently.** They used lists in Padma from Term 3, Week 9.
- [ ] Prepare the physical demo: five cards in a row, numbered 0–4.
- [ ] Know the two errors they will hit: `IndexError` and appending to an undefined name.
- [ ] Write on the board: **counting starts at 0.**

3. Session A — Introduce (40 min)

Show the translation (5 min). Put the Padma list blocks on the projector next to Python.

PADMA	PYTHON
`add (x) to (faults)`	`faults.append(x)`
`item (1) of (faults)`	`faults[0]`
`delete (1) of (faults)`	`del faults[0]`
`length of (faults)`	`len(faults)`
`(faults) contains (x)?`	`x in faults`

> **Script:** "You have built lists before. In Term 3 you kept a fault log — every failure appended to a list, with a timestamp, so the robot could tell you what happened at minute nine. Same idea. Three things are different, and none of them is the idea."

The three differences (say them now, before the demo):

1. **Counting starts at 0.** Not 1. Padma's list monitor shows `1, 2, 3`; Python's first item is `[0]`.
2. **A list is a value.** You can pass it into a function, return it, and put one list inside another.
3. **Two names can point at the same list.** This one will surprise you in a minute.

The first is convention and will cost you a month of debugging. The other two are genuinely new.

Live demo (10 min).

```
readings = [12, 30, 8, 44]
print(readings[0])     # 12
print(readings[3])     # 44
print(len(readings))   # 4
readings.append(19)
print(len(readings))   # 5
```

> **Script:** "Four values, one name. The position is called an index, and the first one is **zero**. Not one. Zero. That is the single thing everyone gets wrong."

Show `readings[4]` → `IndexError`.

Student practice (20 min).

TASK	REQUIREMENT
A	Create a list of four sensor readings and print the first and last
B	Print the length
C	Append two more; print the length again
D	Print every reading with a `for` loop

Task D is where loops and lists meet:

```
for r in readings:
    print(r)
```

Compare (5 min). Ask three students for their Task D.

> **Script:** "Notice the loop knows nothing about how many there are. Add ten readings and the loop still works. That is why lists matter."

4. Session B — Extend (40 min)

Recap (5 min). "What index is the last item in a list of length 4?"

One new element — aliasing (10 min). The one thing Padma could never have taught you.

```
a = [1, 2, 3]
b = a          # NOT a copy — a second name for the same list
b.append(4)
print(a)       # [1, 2, 3, 4]   ← and they will not believe it
```

Then:

```
b = a.copy()
b.append(5)
```

```
print(a)       # [1, 2, 3, 4]   ← copy left a alone
```

> **Script:** "Two names, one list. In Padma a list had exactly one name, so this could never happen to you. Here it happens every week, and the bug is invisible — you change `b` and `a` changes. **If you are going to remember one thing from this week, it is `copy()`.**"

> **Script:** "And the tell: a function takes your list, does something to it, and your list comes back changed. That is not a function misbehaving. That is the function holding the same list you are holding."

One new element — nesting (5 min). Put a list inside a list.

```
grid = [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]]
grid[1][1] = 1
print(grid)
```

> **Script:** "Three rows, three columns. You built this shape in Term 1 — the lotus wall was a loop inside a loop, drawing a 3×3 grid. This is the same grid, stored instead of drawn. Padma lists were flat, so this is genuinely new."

Change the requirement (20 min). Rebuild the Term 3 fault log in Python:

- Start with an empty list `faults = []`
- Ask for three faults with `input()`
- Append each, with a number: `f"1 – {entry}"`
- Print the whole list
- Print how many had the word `wiring` in them

Guard division by zero where relevant: `if len(faults) > 0:`

Exit ticket (5 min). "*Why does `readings[4]` fail on a list of four items?*" and "*what does `b = a` actually do?*"

5. Differentiations

Supporting struggling students

- Tasks A–C only; the loop comes next lesson.
- Physical cards first: point at index 0, 1, 2.
- Pre-write the loop line; they fill in the list name.

Advanced extension

- Find the largest value without using `max()`.
- Reverse the list without `reverse()`.
- Count how many readings are above 20.

WEEK 7

Port the Follower

1. Lesson Info Block

Topic	Bringing Term 3's four-state logic into Python as a simulation
Duration	2 × 40 minutes
Keywords	simulation · state · function · list · return
Mastery statement	*"I can take a program that ran on a robot and run it in text."*
Materials	Computers · the four-state table from Term 2 · the Term 3 follower code on the projector

2. Teacher Preparation Checklist

- [ ] Bring the students' **original four-state tables**. This is the second time they matter.
- [ ] Prepare the target program (below) and test it.
- [ ] Decide: do you inject a fault for Session B, or let them find their own?

3. Session A — Port (40 min)

Show the blocks (5 min). The Padma follower, with `drive` and the four branches.

Live port (10 min). The shape first:

```
def step(left, right):
    if left and right:
        return (50, 50)
    elif right and not left:
        return (50, 20)
    elif left and not right:
        return (20, 50)
    else:
        return (0, 0)
```

```
print(step(True, False))
```

> **Script:** "This function takes the two sensor readings and returns two wheel speeds. No motors — it just *reports* what the robot would do. That is a **simulation**, and it is how you test logic without a robot."

Student port (20 min).

TASK	REQUIREMENT
A	Type the function above and run it for all four inputs
B	Compare the four outputs with your Term 2 table
C	Change speed 50 to 70 and explain what changed
D	Add a fifth case: `(None, None)` meaning "sensor failed"

Task B is the moment: their own handwriting, now executable.

Compare (5 min).

> **Script:** "Anyone whose outputs disagree with their Term 2 table? … You found a bug in a program from two terms ago, using text. That is what this course is for."

4. Session B — Extend (40 min)

Recap (5 min). "What is a simulation, and why is it easier to test than a robot?"

One new element (10 min). Driving the simulation from a **list of states**.

```
states = [(True, True), (True, False), (False, True), (False, False)]

for left, right in states:
    speeds = step(left, right)
    print(f"L={left} R={right} → {speeds}")
```

> **Script:** "One line per state. Four states, four lines of output, in order. Now you can test a whole run without touching anything."

Change the requirement (20 min). They write:

```
def run(states):
    corrections = 0
    lost = 0
    for left, right in states:
        if left and right:
            pass                                    # centred, no correction
        elif left or right:
            corrections += 1
        else:
            lost += 1
    return corrections, lost

course = [(True, True), (True, False), (False, True), (True, True), (False, False)]
print(run(course))
```

> **Script:** "This is the first program in the course that does something the Padma version could not: it *counts its own behaviour* and reports it. A robot cannot tell you how many times it corrected. A simulation can."

Exit ticket (5 min). "*What can this simulation do that KanKaung cannot?*"

5. Differentiations

Supporting struggling students

- Give `step()` complete; they write only the list and the loop.
- Allow `if` chains instead of tuple unpacking.

Advanced extension

- Add a `speed` parameter and recompute corrections at three speeds.
- Return the average correction rate as a percentage.
- Explain why a simulation cannot prove the robot works.

WEEK 8

Read the Error

1. Lesson Info Block

Topic	Tracebacks, exception types, isolating a fault in text
Duration	2 × 40 minutes
Keywords	traceback · exception · TypeError · SyntaxError · isolate · hypothesis
Mastery statement	*"The error message tells me what is wrong. I just have to read it."*
Materials	Computers · six deliberately broken programs, pre-typed and saved

2. Teacher Preparation Checklist

- [ ] Prepare **six broken files**. Test each one — it must produce the intended error.
- [ ] Print the error card (below) at A5, one per desk.
- [ ] Write on the board: **last line first.**
- [ ] Accept that this lesson feels slow and that is correct.

3. Session A — Teach the Traceback (40 min)

Show the problem (5 min).

> **Script:** "Up to now, when something broke, you asked me. From today, you ask the machine. It always answers. Most people never read the answer."

Live demo (10 min). Deliberately break a program and read the traceback aloud, **bottom to top**.

```
power = "100"
power = power - 4
```

```
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

> **Script:** "Read the last line first — it names the error. Then find the line number above it. Then ask: what did I expect that line to be? The message told me everything."

The six broken programs (20 min). Teams of two, one file each, rotate every three minutes.

#	FILE	ERROR	WHAT IT TEACHES
1	Missing closing bracket	`SyntaxError`	The error is *before* where you think
2	`if x = 5:`	`SyntaxError`	`=` vs `==`
3	`print("Speed: " + 50)`	`TypeError`	Text + number
4	`input()` used without `int()`	`TypeError`	Input is text
5	`readings[5]` on 4 items	`IndexError`	Zero-index and bounds
6	Function called with 1 argument, defined with 2	`TypeError`	Signatures

Compare (5 min).

> **Script:** "Which error took the longest? Number one, usually — because the message points at the *next* line, not the broken one. Python found the problem when it reached the end of the statement."

4. Session B — Break Your Own (40 min)

Recap (5 min). "Which line of a traceback do you read first?"

One new element (10 min). The six-step protocol in text.

STEP	PADMA	PYTHON
1 Predict	Say what should happen	Same
2 Run	Click the flag	Run the file
3 Observe	Say exactly what is wrong	**Read the last line**
4 Isolate	Delete a block	Comment out a line with `#`
5 Fix	Change one block	Change one line
6 Re-test	Run again	Run again

> **Script:** "`#` at the start of a line makes Python ignore it. That is how you remove something without deleting it — and it is how you find out which half is broken."

Change the requirement (20 min).

- Each pair writes **one** new broken program that produces a specific error

- They swap with another pair, who must name the error type and the fix **without running it**
- Then they run it and check

Exit ticket (5 min). "*What are the three things you look at in a traceback?*"

5. Differentiations

Supporting struggling students

- Three files instead of six.
- Give the error card with the six types listed.
- Let them read the traceback aloud to a partner before writing anything.

Advanced extension

- Write a `try` / `except` around the input conversion so a bad value does not crash the program.
- Explain why a `SyntaxError` has no line number in the same place as a `TypeError`.

WEEK 9

Algorithms

1. Lesson Info Block

Topic	Linear search, bubble sort, writing pseudocode
Duration	2 × 40 minutes
Keywords	algorithm · search · sort · pseudocode · trace
Mastery statement	*"I can describe a procedure precisely enough to follow it by hand."*
Materials	Computers · a set of 8 number cards per pair · pseudocode reference sheet

2. Teacher Preparation Checklist

- [ ] Prepare the card sort: eight cards, unsorted.
- [ ] Print the pseudocode reference (below).
- [ ] Have the bubble sort trace ready on paper — you will do it by hand first.

3. Session A — Search and Sort (40 min)

Unplugged first (5 min). Eight cards face down in a row.

> **Script:** "Find the number 7. Turn cards over one at a time. How many did you turn? That is a linear search, and now you know its cost."

Live code (10 min).

```
def linear_search(items, target):
    for i in range(len(items)):
        if items[i] == target:
            return i
    return -1

print(linear_search([4, 9, 2, 7, 5], 7))
```

> **Script:** "Notice the `return -1` at the end — outside the loop. If it were inside, it would give up after one item. Position of the `return` is the algorithm."

Bubble sort by hand (10 min). On the board with six numbers. One pass only.

> **Script:** "Compare neighbours. Swap if out of order. Move right. One pass moves the biggest number to the end. Now repeat — and you can shorten each pass, because the end is already sorted."

Student practice (15 min).

```
def bubble(nums):
    n = len(nums)
    for i in range(n - 1):
        for j in range(n - 1 - i):
            if nums[j] > nums[j + 1]:
                nums[j], nums[j + 1] = nums[j + 1], nums[j]
    return nums

print(bubble([5, 2, 9, 1, 7]))
```

Tasks: run it; add a print inside the outer loop to show the list after each pass; count the comparisons.

4. Session B — Three Notations (40 min)

Recap (5 min). "Why is the inner loop's range shorter each time?"

One new element (10 min). The same algorithm in three notations — flowchart, pseudocode, Python.

Put all three on the board side by side for `linear_search`:

```
FLOWCHART                    PSEUDOCODE                  PYTHON

 ┌──────────┐               FUNCTION search(items,t)    def search(items, t):
 │  START   │                   FOR i FROM 0 TO             for i in range(len(items)):
 └──────┬───┘                       LENGTH(items)-1              if items[i] == t:
      ▼                         IF items[i] = t THEN             return i
 ◇ any items? ◇ ──NO──► stop          RETURN i                 return -1
      │ YES                     ENDIF
      ▼                     NEXT i
 [ get next ]               RETURN -1
      │                    ENDFUNCTION
      └──► back to ◇
```

> **Script:** "Three pictures of one idea. The flowchart is for thinking. The pseudocode is for marks. The Python is for running. **Same algorithm, three costumes.**"

> **Script:** "Notice which one has no syntax errors. The flowchart. You can be wrong about a lot of things in Python and the flowchart will never care. That is why you draw before you type."

Change the requirement (20 min). Pick one of these and do all three notations:

- Find the largest value in a list
- Count how many items are above 20
- Reverse the list into a new list

Then: **write the flowchart, write the pseudocode, write the Python, run the Python, and check all three agree.**

Exit ticket (5 min). "*Which notation would you use to explain an algorithm to a marker, and why?*"

5. Differentiations

Supporting struggling students

- Linear search only; bubble sort next term.
- Give the pseudocode skeleton with keywords filled in.

Advanced extension

- Count comparisons for a list of 5 versus 10 — what is the pattern?
- Explain why bubble sort is slow.
- Write pseudocode for a binary search on a *sorted* list.

WEEK 10

Data Representation

1. Lesson Info Block

Topic	Binary, denary, hexadecimal, characters, sound and images
Duration	2 × 40 minutes
Keywords	binary · bit · byte · denary · hexadecimal · ASCII · sampling
Mastery statement	*"I can convert between number systems and explain how a computer stores text, sound and pictures."*
Materials	Computers · 8 cards marked 128, 64, 32, 16, 8, 4, 2, 1 · hex reference

2. Teacher Preparation Checklist

- [ ] Make the eight binary cards. Physical cards beat a diagram, every time.
- [ ] Practise 173 → binary until you can do it in your sleep: **10101101**.
- [ ] Prepare three conversion examples at three difficulties.
- [ ] Note: this week has **no ported Padma program**. It is new subject matter, and students who expect code will be thrown.

3. Session A — Binary and Hex (40 min)

The cards (5 min). Eight students hold 128, 64, 32, 16, 8, 4, 2, 1. Face up or down.

> **Script:** "Face down means 0, face up means 1. Add up whoever is face up. What number do we have? … Now make 173. Which cards must be face up?"

Answer: 128 + 32 + 8 + 4 + 1 = 173 → `10101101`.

Live demo (10 min).

```
n = 173
print(bin(n))    # 0b10101101
print(hex(n))    # 0xad
```

> **Script:** "Python will do it for you — and you must be able to do it by hand, because the exam will not give you Python. The `0b` and `0x` are labels, not part of the number."

Student practice (20 min).

TASK	REQUIREMENT
A	Convert 173, 60 and 255 to binary by hand; check with `bin()`
B	Convert 10110100 to denary
C	Add two 4-bit binary numbers
D	Convert 173 to hex by hand: split into two nibbles

Compare (5 min). Task D.

> **Script:** "Eight bits is one byte. Split it in half: four bits, four bits. `1010` is A, `1101` is D. So 173 is AD. Hex is not a new number system — it is a way of writing bytes short."

4. Session B — Representation of Everything Else (40 min)

Recap (5 min). "How many bits in a byte? How many different values can a byte hold?"

Characters and sound (10 min).

```
print(ord("A"))     # 65
print(chr(97))      # a
```

> **Script:** "Text is numbers with an agreed mapping. A is 65. Not because A is 65th — because someone decided, and everyone agreed. Sound and images are the same trick: numbers, sampled finely enough that our eyes and ears are fooled."

Then the two ideas that earn marks: **resolution** (how many samples) and **bit depth** (how precise each sample).

Change the requirement (20 min).

- Convert your own name to ASCII values by hand (use `ord` to check)
- Explain why a 44,100 Hz sample rate is enough for human hearing
- Explain why a 100×100 image at 8-bit colour is smaller than 1000×1000 at 24-bit

Exit ticket (5 min). "*Why can a computer store a photograph as numbers?*"

5. Differentiations

Supporting struggling students

- Four bits only (0–15) this week.
- Give the conversion table 0–15 pre-filled.

Advanced extension

- Two's complement: why −1 is `11111111`.
- Calculate the file size of a 3-minute stereo recording.
- Explain lossy vs lossless in one sentence.

WEEK 11

Networks and Security

1. Lesson Info Block

Topic	Topologies, protocols, threats, mitigation
Duration	2 × 40 minutes
Keywords	network · LAN · topologies · protocol · IP · encryption · firewall
Mastery statement	*"I can describe how machines talk, and how that conversation is attacked."*
Materials	Computers · string or rope for a topology demo · a stack of cards for the packet game

2. Teacher Preparation Checklist

- [ ] This week is **mostly discussion**. Warn the class in advance; some will be disappointed.
- [ ] Prepare the topology demo with string: four students in a ring, then in a star.
- [ ] Know the four threats you will teach: phishing, malware, weak passwords, unpatched software.
- [ ] Prepare one real-world story from your own country — local relevance beats any textbook.

3. Session A — How Networks Work (40 min)

Topology with string (10 min). Four students hold string in a ring, then re-form into a star around one.

> **Script:** "Ring: one cut breaks everything. Star: one cable breaks, three people keep working — but if the centre dies, nobody works. Every topology is a trade-off between **resilience** and **cost**."

Discussion (10 min). LAN vs WAN. What is in the room, what is not.

Then protocols, briefly: **IP** is the address, **TCP** is the delivery, **HTTP** is the language.

Student task (15 min).

- Draw your school's network on paper: devices, switch, router, the connection out
- Label what is a LAN and what is not
- Mark the single point of failure

Compare (5 min). Two drawings on the board.

4. Session B — Security (40 min)

Recap (5 min). "Where is the weakest point in your school's network?"

Discussion (10 min). The four threats, each with a mitigation:

THREAT	HOW IT WORKS	MITIGATION
Phishing	Pretends to be someone you trust	Verify the sender; never click a link to log in
Malware	Arrives with something you wanted	Updates; no unofficial installers
Weak passwords	Guessed or reused	Length over complexity; a password manager
Unpatched software	A known hole left open	Automatic updates

> **Script:** "Notice that three of the four are *people* problems, not technical ones. That is why security is a lesson in a computing course and not just a job for the IT department."

Change the requirement (20 min).

- Write a five-point security policy for the school computer room
- Justify each point in one sentence, naming the threat it addresses
- Rank them by how likely a student is to break them

Exit ticket (5 min). "*Name one threat and its mitigation, in one sentence.*"

5. Differentiations

Supporting struggling students

- Three threats instead of four.
- Give the policy as sentence starters.

Advanced extension

- Explain encryption to a 10-year-old in three sentences.
- Why is public Wi-Fi risky even with a password?
- What does a firewall actually inspect?

WEEK 12

Mock and Practical

1. Lesson Info Block

Topic	A timed theory paper and a timed programming task
Duration	2 × 40 minutes
Keywords	assessment · trace · practical · time management · readiness
Mastery statement	*"I can work under time, alone, without help."*
Materials	Printed theory paper (30 marks) · practical task sheet · computers · a visible timer

2. Teacher Preparation Checklist

- [ ] Print the theory paper and the practical sheet, one per student.
- [ ] **Do not teach this week.** Assess it.
- [ ] Mark both papers before the next session; return with the readiness statement.
- [ ] Prepare the readiness statement template (below).

3. Session A — Theory (40 min)

Instructions (5 min). 30 minutes, 30 marks, no notes, no help.

> **Script:** "This is not a test of you. It is a test of the course. If half the class fails a question, I taught it badly — and I will look at that before I look at your mark."

The paper (30 min).

Q	TOPIC	MARKS
1	Convert 173 to binary and to hex	4
2	Convert `10110100` to denary	2
3	Explain why `"5" + 5` is an error	3
4	Trace a 10-line program and state the output	5
5	Write pseudocode for finding the largest value in a list	6
6	Name three network topologies and one advantage of each	3
7	Name two threats and a mitigation for each	4
8	Explain why a byte can hold 256 values	3

Collect (5 min).

4. Session B — Practical (40 min)

Instructions (5 min).

> **Script:** "Forty-five minutes is not enough if you plan to type it twice. Write the function first, test it on three inputs, then finish the rest. A program that runs on three cases beats a program that is beautiful and runs on none."

The task (35 min).

> **Specification.** The greenhouse sensors report a sequence of readings, each a pair of booleans: `(left, right)`, where `True` means that sensor saw the line.
>
> Write a program that:
>
> 1. takes a list of readings,
> 2. counts how many times the robot was **centred** (both true),
> 3. counts how many times it had to **correct** (exactly one true),
> 4. counts how many times it was **lost** (both false),
> 5. prints all three counts and a one-line verdict.
>
> A run is **good** if it was lost fewer than 3 times.

Marking: function structure (4) · correct counting (6) · verdict (2) · runs without error (3). **15 marks.**

5. Return and readiness

Next session, return both papers with a **readiness statement**, one page per student:

Trace and explain	___ / 5
Write pseudocode	___ / 6
Theory	___ / 19
Practical	___ / 15
Recommendation	☐ Ready for iGCSE ☐ One more term of practice ☐ Not yet

> **Say this out loud, to the class:** "A 'not yet' is not a judgement of you. It means you would struggle in a two-year course right now, and I would rather tell you that here than have you find out in month six."

The 12 weeks at a glance

WK	SESSION A — PORT	SESSION B — EXTEND	DESCRIPTOR
1	print, quotes, running a file	indentation as structure	PL1
2	variables and assignment	`input()`, `int()`, f-strings	PL2
3	if/elif/else	the four-state table in Python	PL3
4	for and range	while, and the infinite loop	PL4
5	def and parameters	return	PL5
6	lists, index, append	loop with index	PL6
7	port the follower	run a list of states	TR2
8	tracebacks	break your own program	DG1–3
9	linear search, bubble sort	pseudocode	TH5
10	binary and hex	characters, sound, images	TH1–2
11	topologies, protocols	threats and mitigation	TH3–4

WK	SESSION A — PORT	SESSION B — EXTEND	DESCRIPTOR
12	theory paper	practical task	EX1–3

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

1. Run a Python file and read its output.
2. Explain why indentation matters.
3. Convert a value with `int()` after `input()`.
4. Write an `if` / `elif` / `else` chain from a table.
5. Write a `for` loop and a `while` loop, and say when each is right.
6. Define a function with a parameter and a `return`.
7. Create a list, index it, append to it, loop over it.
8. Read a traceback and name the error type.
9. Convert 173 to binary and hex, by hand.
10. Write pseudocode for a simple algorithm.
