# How a Block Program Actually Runs (read this first)

This single idea explains almost every confusing thing students will hit in these assignments. Five minutes here will save you an hour in class.

## The satellite re-reads the whole program every second

A Zero Robotics graphical project has **two pages** of blocks:

| Page | When it runs |
|---|---|
| **init** | once, at the moment the match starts |
| **main** ("loop") | **once per second, every second, for the entire 3-minute match** |

When the match runs, the satellite does this, 180 times in a row:

> tick: read the **main** page from top to bottom, do every block on it, in order, all within that one second… then wait for the next tick and **do the whole page again from the top.**

The blocks on the main page are **not a story that plays out over time**. They are a **checklist the satellite re-reads every second**. The program never "moves on" past a block; next second, that block runs again.

## Why this trips students up

Students arrange blocks like a recipe: go here, then take a picture, then go there, and expect the satellite to do step 1, finish it, then step 2. Instead, **all three blocks run every second**, immediately, one after another, 180 times. The result on screen: the satellite obeys only the last movement block, or fires the camera instantly before it has aimed, and the students say "it skipped my blocks."

It didn't skip anything: it ran everything, every second.

## The two tools that make multi-step plans work

1. **A variable that remembers which step we're on.** Variables keep their value between ticks. A variable named `step` declared on the init page is the program's memory.
2. **`if` blocks that check the step every tick.** Each tick, the program asks "which step am I on?" and only does that step's blocks. Another `if` asks "is this step finished?" (e.g. am I within 5 cm of the target?) and, when it is, changes `step`, so the next tick does the next step.

That pattern (**remember the step, act on the step, advance the step when its goal is met**) is the heart of this whole curriculum (engineers call it a state machine). Assignments 1–4 build it up one piece at a time; assignments 5–7 use it to play the game.

## Three quick facts that follow from "everything runs every second"

- **Repeating a movement block is harmless.** `setPos 0.4, 0.4, 0` every second just keeps the destination set; the satellite flies there and stops. One block really can be a whole flight plan.
- **Two movement blocks in a row is NOT two stops.** Both run within the same second; the second one overwrites the first. The satellite flies only to the last target.
- **A `repeat 10 times` loop block does not last 10 seconds.** It repeats within one tick. Time passes only between ticks, so anything meant to unfold over match time needs the variable + `if` pattern instead.

## What the generated C looks like

The IDE's "C Code" toggle shows what the blocks become. The init page becomes `void init() { … }` and the main page becomes `void loop() { … }`; the simulator calls `init()` once and then calls `loop()` once per second. Block program and C program are two views of the same thing, which is also how these materials let you verify a solution: the reference C in each assignment is exactly what the finished block layout generates.