LESSON 94

Developing the Simulation Algorithms

240 minutes (6 periods) | Applied Learning Tasks (ALTs) | Outcome 1.9 | Outcome 2.5 | Outcome 2.6 | Outcome 2.7 | Outcome 2.20 | Outcome 3.8 | Outcome 3.9

WHAT THIS LESSON DOES

In this lesson, you'll build simulation algorithms using Python, starting with pseudocode templates. Follow steps to design iterative processes, implement agent-based models, test modifications, debug issues, and document your progress to understand emergent behaviours in simulations.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.9	use modelling and simulation in relevant situations
2.5	use pseudo code to outline the functionality of an algorithm
2.6	construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement
2.7	implement algorithms using a programming language to solve a range of problems
2.20	identify and fix/debug warnings and errors in computer code and modify as required
3.8	develop a model that will allow different scenarios to be tested
3.9	analyse and interpret the outcome of simulations both before and after modifications have been made

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the fundamental concepts of simulation algorithms, including iterative processes and agent-based modelling.
- Design adaptable pseudocode templates for various simulation scenarios.
- Implement simulation algorithms in Python to model dynamic systems.
- Analyse the impact of parameter modifications on simulation outcomes.
- Develop debugging skills to identify and resolve issues in simulation code.

BEFORE THE LESSON

- Run both example programs yourself first: the population growth loop and the random-walker agent model. Confirm Python and the random module work on the room machines.
- Have the two starter snippets saved where every student can open them, so nobody loses time retyping the Agent class.
- This is a long session across two periods. Decide where the natural break falls, ideally after Python implementation and before testing modifications.

Developing the Simulation Algorithms

HOW THE LESSON RUNS

Introduction
Understanding Simulation Algorithms
Designing Coding Templates
Implementing in Python
Testing Modifications
Debugging Activities
Document Your Work

TEACHING MOVES

- 2. Understanding Simulation Algorithms. Draw the time-step idea on the board: one loop pass equals one unit of time. Contrast the two models plainly. Iterative tracks totals, agent-based tracks individuals whose simple rules add up to emergent behaviour like a traffic jam.
- 3. Designing Coding Templates. Keep templates as pseudocode, not Python, at this stage. Point out the shared skeleton: initialise, loop, update, store. Emerging students copy the template; ask others to name what changes between the iterative and agent versions.
- 4. Implementing in Python. Have students run the growth loop before touching the agent code. Once numbers print, move to the Agent class. Explain that each agent is an object holding its own position, which is why they need a list of them.
- 5. Testing Modifications. Insist they record the base result before changing anything. Then change one parameter at a time. Ask them to predict the outcome before running, so testing becomes reasoning rather than random tweaking.
- 6. Debugging Activities. Run the infinite-loop example live so they watch it hang, then fix it together. Push print-statement tracing as the first tool, not a last resort. Have them talk through possible causes before editing.
- 7. Document Your Work. Have students note the parameters they changed, what happened, and one bug they fixed. This record feeds the refinement and visualisation stage and is good evidence practice for their own project work.

WHAT TO WATCH FOR

- Students think changing growth_rate randomly until output looks reasonable counts as testing. Require a written prediction and a recorded baseline before each change. Testing means comparing against an expectation, not hunting for a nice-looking number.
- Students expect the random walkers to end up in the same place each run and treat the varying output as a bug. Point out random.choice makes each run different by design. Ask them to run it several times and describe the spread rather than a single answer.
- A while loop with no decrement is assumed to eventually stop on its own. Show the loop hanging, then trace why the condition never becomes false. Reinforce that every while loop needs something inside it that moves toward the exit.

DIFFERENTIATION

- Emerging: Give them the working iterative growth loop and ask only that they change one number and describe the effect, before any agent code.
- Developing: Ask them to add a death_rate to the population model themselves, then explain in words why the curve changes shape.
- Proficient: Challenge them to add a random event to the agent model, such as agents that stop or reverse under a condition, and describe the emergent behaviour it produces as evidence for their project write-up.