LESSON 74

Modelling in Design

120 minutes (3 periods) | Software Engineering and Testing | Outcome 1.9 | Outcome 2.1 | Outcome 2.4 | Outcome 1.22 | Outcome 3.8 | Outcome 3.9

WHAT THIS LESSON DOES

In this lesson, you'll explore the core ideas of simplifying and solving real-world problems through programming. You'll gain practical experience by creating Python programs that apply abstraction, modelling, and simulation concepts to tackle design challenges effectively.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.9	use modelling and simulation in relevant situations
2.1	use abstraction to describe systems and to explain the relationship between wholes and parts
2.4	illustrate examples of abstract models
1.22	read, write, test, and modify computer programs
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

- Grasp the concept of abstraction to simplify complex systems by focusing on essential features.
- Understand modelling as a method to create simplified representations of real-world systems for analysis.
- Learn the principles of simulation to imitate system behaviours and test scenarios over time.
- Develop skills in applying abstraction, modelling, and simulation through Python programming tasks.
- Integrate these concepts into a cohesive project to solve real-world problems using code.

BEFORE THE LESSON

- Test Python and VS Code on a student machine, and confirm the random library imports and runs, before the class arrives.
- Decide where students will create the ModellingLesson folder and make sure they can save files there.
- Have the three code snippets ready to paste or display, since students copy each one in full before running it.

Modelling in Design

HOW THE LESSON RUNS

Introduction
Understanding Abstraction
Task: Abstraction
Understanding Modelling
Task: Modelling
Understanding Simulation
Task: Simulation
Build Your Own Project
Wrap Up

TEACHING MOVES

- 2. Understanding Abstraction. Use the car example concretely: ask what a driver needs to know versus what an engineer needs. Get students to name details they would ignore before you give them the definition.
- 3. Task: Abstraction. Point out that the class hides how hunger changes and only exposes methods to interact. Have students run the file before they read the internals, so they feel the hiding first.
- 4. Understanding Modelling. Stress that a model keeps the essential features from abstraction and uses them to predict. A map and a budget are strong everyday anchors here.
- 5. Task: Modelling. Have students change the temperature and rain values and predict the forecast before running, so the model is something they reason about, not just execute.
- 6. Understanding Simulation. Draw the distinction sharply: a model is the representation, a simulation runs it over time. Traffic lights is the given example, keep it concrete.
- 7. Task: Simulation. Run the dice program twice so students see the output differs each time. Ask why, then connect randomness to what makes it a simulation rather than a fixed calculation.
- 8. Build Your Own Project. Hold the 50 to 100 line limit firm and require each of the three concepts be identifiable. Ask students to point to the abstraction, the model and the simulation in their own code.

WHAT TO WATCH FOR

- Students treat modelling and simulation as the same word, since both tasks produce output. Anchor on the difference: the weather function is a model, running the dice loop over ten rolls is a simulation. One represents, the other runs over time.
- Students think the dice program is broken because it gives different numbers each run. Confirm this is correct and intended. Randomness is the point of the simulation; a fixed result would not imitate real rolling.
- Students believe abstraction means writing less code, rather than hiding detail behind a clean interface. Show that the Pet class is more lines than a bare variable, but the code that uses it is simpler. Abstraction manages complexity, it does not always shorten.

DIFFERENTIATION

- Emerging: If VS Code or the folder is fighting a student, get them running the pasted code first and reasoning about it second. Let them complete the three set tasks and skip the open project rather than start it and stall.
- Developing: Ask them to explain aloud which method of the Pet class hides what, before moving on. For the project, give them a fixed theme so they spend effort on the three concepts, not on choosing a scenario.
- Proficient: Challenge them to run the dice simulation hundreds of times and report the average, moving toward the Higher Level treatment of simulation and randomness. Have them document where abstraction, modelling and simulation appear in their project as evidence they could reuse in an Applied Learning Task.