Lesson 3: At a crossroads

**Enjoyed teaching these lessons? Found a mistake? Share feedback at** [**the-cc.io/feedback**](http://the-cc.io/feedback)**.**

## Introduction

This lesson introduces selection and randomness. These are two features that will allow learners to develop programs with a very diverse range of behaviours.

Learners will revisit some of the programs that they have encountered in previous lessons and extend them into more versatile programs that use selection. They will develop a simple number guessing game, which will eventually include randomness.

In this lesson, selection is limited to binary choices, i.e. there are always only two possible branches. Multiple branches and nested structures will be introduced later on.

## Learning objectives

* Use relational operators to form logical expressions
* Use binary selection (if, else statements) to control the flow of program execution
* Generate and use random integers

## Key vocabulary

Selection, relational (or comparison) operators, logical (or Boolean) expressions, conditions, randomness, execution, walk-through

## Preparation

**Subject knowledge:**

* You will need to be familiar with using a Python IDE.
* You will need to be able to locate and correct syntax errors in Python programs.
* You will need to be comfortable with the use of output, input, and assignment in Python, including arithmetic input.
* You will need to be comfortable with the use of arithmetic and relational operators, and arithmetic and logical expressions (conditions).
* You will need to be comfortable with the use of selection in Python programs.
* You will need to be aware of common misconceptions that you may encounter with novice learners. See the [common misconceptions](#_thliz98poo7u) in the ‘Notes on pedagogy’ section for a list of misconceptions relevant to this lesson.

**You will need:**

* Slides — note that some slides contain animations
* Activities:
  + Selection in Python: [Live coding starting point](https://the-cc.io/py-greeting-30) (the-cc.io/py-greeting-30)
  + Practise using selection:
    - Worksheet
    - Python code: [Film critic — starting point](https://the-cc.io/py-critic-30) (the-cc.io/py-critic-30)
    - Python code: [Lucky number — starting point](https://the-cc.io/py-lucky-30) (the-cc.io/py-lucky-30)
  + Randomness:
    - Python code: [Lucky number — live coding starting point](https://the-cc.io/py-lucky-31) (the-cc.io/py-lucky-31)
    - Python code: [Random lucky number — live coding](https://the-cc.io/py-lucky-32) (the-cc.io/py-lucky-32)
* A Python interpreter and IDE — we suggest using the Mu editor ([codewith.mu](https://codewith.mu/)), or an online environment such as [Repl.it](https://repl.it/)

**You may need:**

* Additional Python code for the ‘Selection in Python’ activity:
  + Live coding: [Greeting — complete program](https://the-cc.io/py-greeting-31) (the-cc.io/py-greeting-31)
* Additional Python code for the ‘Practice using selection’ activity:
  + [Film critic — solution](https://the-cc.io/py-critic-31) (the-cc.io/py-critic-31)
  + [Lucky number — solution](https://the-cc.io/py-lucky-31) (the-cc.io/py-lucky-31)
  + Explorer task: [Eligible to vote — solution](https://the-cc.io/py-vote-31) (the-cc.io/py-vote-31)
* Additional Python code for the ‘Randomness’ activity:
  + [Dice roll](https://the-cc.io/py-dice-31) (the-cc.io/py-dice-31)
* Python cheat sheets on operators and expressions, selection, and using modules

## Assessment opportunities

Closed-formed questions throughout the lesson provide a means for quick formative assessment on specific concepts. You can also assess learners’ answers to the worksheets. In addition, you can assess learners through observation, for example, by assessing how learners interact through pair programming and collaborate to solve problems.

At a glance

| **Starter activity**  5 mins | **Something missing**  Revisit three of the sequential programs that learners have encountered in the previous lessons and describe ‘what if’ scenarios that would involve extending each program to execute different instructions depending on the input. Discuss these scenarios and remind learners of selection. |
| --- | --- |
| **Activity 1**  12 mins | **Selection in Python**  Show the general form of the if statement in Python and explain when it is necessary. Use live coding to build a first example that employs selection, along with your learners. |
| **Activity 2**  20 mins | **Practise using selection**  Ask learners to carry out the ‘Film critic’ and ‘Lucky number’ tasks in the ‘Practise using selection’ worksheet. They can use the code in the worked example included in the worksheet as a point of reference. |
| **Activity 3**  8 mins  3 mins labels  5 mins walk-through | **Exploring the number guessing game**  Display the code for the number guessing game, and identify and label the distinct sections in the program. Perform a walk-through of the part of the program that contains the selection structure, step by step, along with your learners. |
| **Activity 4**  10 mins  4 mins live coding  2 mins questions  4 mins task | **Randomness**  Use live coding to extend the number guessing game to pick a random lucky number. Use a couple of multiple choice questions on generating random integers for a dice rolling program and allow learners some time to type the code and execute it a few times, so that they can see how a random number is generated each time. |
| **Activity 5**  10 mins | **Plenary**  Ask a few closed-form questions (true/false, multiple choice) to assess how comfortable learners are with randomness and the selection structure. |

## 

## Outline plan

Please note that the slide deck labels the activities in the top right-hand corner to help you navigate the lesson.

*\*Timings are rough guides*

| **Starter activity**  (Slides 2–3)  5 mins | **Something missing**  First, revisit one of the sequential programs that learners encountered in the previous lessons (slide 2). Display the code and stress the fact that it’s always the same instructions that are executed, regardless of the input.  Then, describe a ‘what if’ scenario that would involve extending the program to execute different instructions, depending on the input. For example: “This program greets every user by name. What if we wanted the program to recognise one particular name and treat it differently?”  Click through the animations on slide 2 to display two additional programs from the previous lessons, along with a ‘what if’ scenario for each one. Discuss these scenarios with your learners. Ask them to describe how the user might interact with the program (e.g. possible inputs and the corresponding outputs) and how the program might work internally. If they use any important keywords, such as **if**, **else**, **otherwise**, **check**, **condition**, **choose**, **depend**, etc., write them on the board.  If it hasn’t come up already, ask learners how this sort of problem was handled in Scratch. Ask them what sort of block was used to perform different actions depending on a condition. They should be able to recall or describe the control blocks associated with selection, and this will provide useful context.  To conclude, remind learners of the concept of **selection**, a control structure that involves checking conditions and selecting accordingly which path to follow through a program (slide 3). |
| --- | --- |
| **Activity 1** (Slides 5–11)  12 mins  2 mins intro  10 mins live coding | **Selection in Python**  Display slide 5 to show the general form of the if statement in Python and compare it with the two corresponding selection blocks available in Scratch, to enable learners to make associations.  Display slide 6 to show flow chart outlines next to the Python code, to visually emphasise that this form of selection describes two possible paths that a program **might** follow during execution. Move your hands along the possible paths to illustrate that the branches are mutually exclusive and that only one of the possible paths will be followed during execution.  Remind learners that selection is necessary in a program whenever the program must execute different actions in different circumstances.  **Live coding: Extending a program to use selection**  First, pair learners, as they will be working using **pair programming**.  Use **live coding** to build the first example of using selection, while your learners follow. Start with the existing program that greets the user by name and extend it.  Learners should already be familiar with selection from the Y7 Programming units, but you can use slide 9 as a recap in the context of Python. Focus your explanations mainly on the structure of the if statement (the condition, the if block, the else block) and how this will work during program execution.  After writing your program, turn your attention to the finer points of the if statement syntax in Python (e.g. no capitalisation of the keywords if and else, the colon, the equality relational operator, and — crucially — indentation). Display slide 10 to highlight the ‘syntactic pitfalls’ of the if statement in Python.  Ask learners to run and test their programs. While they are doing that, hand out the ‘Practise using selection’worksheet. The worked example on the first page contains the correct version of the code. There is also a checklist; instruct learners to use it, in order to check for and correct possible syntax errors.  Briefly read through the reference slide (slide 11) that contains the relational operators (comparisons) that can be used in conditions. |
| **Activity 2** (Slides 12–14)  20 mins  18 mins worksheet  2 mins solutions | **Practise using selection**  Ask learners to carry out the ‘Film critic’ and ‘Lucky number’ tasks in the ‘Practise using selection’ worksheet. They can use the code in the worked example included in the worksheet as a point of reference.  Remind learners to switch between the driver and navigator roles as they move from one task to the other.  If necessary, present and very briefly discuss example solutions for the worksheet tasks (slides 13 and 14). The solution for the ‘Lucky number’ task will serve as a bridge to the next activity. |
| **Activity 3** (Slides 15–17)  8 mins  3 mins labels  5 mins walk-through | **Exploring the number guessing game**  **Subgoal labelling**  Display slide 15, which shows the code for the number guessing game, then click once to activate the animation that will identify the distinct sections in the program. At first, do not describe what each section does. Ask learners to suggest **labels** for these sections that summarise their function. Then, click through the animations to provide examples of what these labels might look like for these particular sections.  **Executing the program**  Use slides 16 and 17 to perform a **walk-through** of the part of the program that contains the selection structure, step by step, while your learners follow. Again, this will help them develop their model of a **notional machine**: it will help them to understand how instructions are executed and, in this case, the ‘mechanics’ of the selection structure.  Perform the walk-through twice: first, demonstrate the case where the user guesses the lucky number (slide 16), and second, demonstrate the case where they don’t (slide 17). This will illustrate how the program specifies **two different paths** through it, but also how only **one of these paths** will be followed during program execution. This is worth highlighting to your learners.  The walk-through needs to be as interactive as possible. In each step, ask learners how the variable state will be affected or what the output will be, before you display the answer on the slides. Most importantly, ask what is the **next instruction** to be executed now that a selection structure is involved.  While you perform the walk-through, highlight the following subtle points around **indentation**:   * The instruction that reveals the lucky number is indented, which means that it belongs to the else block and will only be executed if the user hasn’t guessed the number * The instruction that says goodbye to the user is not indented, which means that it lies outside the selection structure and will always be executed, regardless of the outcome of the game   **Note:** An alternative online tool for step-by-step execution of Python programs and visualisation of program state is [Python Tutor](http://pythontutor.com/visualize.html) (pythontutor.com/visualize.html). |
| **Activity 4**  (Slides 18–22)  10 mins  4 mins live coding  2 mins questions  4 mins task | **Randomness**  Keep displaying the code for the number guessing game (slide 18). Describe a ‘what if’ scenario that would involve extending the program to pick a random lucky number every time it is executed. Use **live coding** to extend the program, while your learners follow.  In your program, insert a print statement to display the lucky number right after it is generated. This will allow you to see the lucky number and test that your program behaves correctly. It will also model a useful testing/debugging method for your learners. Remember to remove the line you added after you have finished testing.  Explain that in order to generate a random integer, a function from a **module** needs to be imported. Use slide 19 to briefly describe what modules are and explain how a random integer can be generated.  Ask a couple of multiple choice questions on generating random integers for a dice rolling program to assess how comfortable learners are with this new idea (slides 20 and 21).  Allow learners some time to type the code for the dice rolling program and execute it a few times, so that they can see how a random number is generated each time (slide 22). |
| **Activity 5**  (Slides 23–25)  10 mins | **Plenary**  Ask a few closed-form questions (true/false, multiple choice) to assess how comfortable learners are with randomness and the selection structure.  It is important to discuss these questions, so make sure that the previous activities do not overrun. |

## Notes on pedagogy

### Common misconceptions

Throughout these lessons, you should be aware of the common misconceptions that may arise with novice programmers. The misconceptions below are a selection from Appendix A of Juha Sorva’s PhD thesis, *Visual Program Simulation in Introductory Programming Education*, and are relevant to this lesson. The list will help you avoid using any examples or explanations that may give rise to these misconceptions, and it will also help you spot them, should they arise with your learners.

**M1, 2, 7:** The computer knows or is able to deduce the intention of the program or of a piece of code, and acts accordingly. The machine understands English.

**M4:** The system does not allow unreasonable operations.

**M6:** Difficulties with telling apart the static and dynamic aspects of programs.

**M10:** Variables always receive a particular default value upon creation.

**M8:** Magical parallelism: several lines of a (simple non-concurrent) program can be simultaneously active or known.

**M36:** All statements of a program get executed at least once.

**M35:** Print statements are always executed, irrespective of branching statements.

**M24:** Code after if statement is not executed if the then clause is.

**M25:** if statement gets executed as soon as its condition becomes true.

**M26:** A false condition ends program if no else branch.

**M27, 29:** Both then and else branches are executed. Using else is optional (the next statement is always the else branch).

**M28:** The then branch is always executed.

**M161:** Boolean values are just something used in conditionals and not data comparable to numbers or strings.

### Flow chart outlines

Throughout these lessons, flow chart outlines are used to illustrate control flow. These outlines are minimal, with no code included in the shapes. They may be arranged in a slightly different layout than usual, to directly **reflect the structure of the program text**.

Learners will not be asked to create flow charts or interpret algorithms represented in this fashion. Flow charts will only be provided alongside Python programs, to reinforce the correct model of program execution. When performing walk-throughs of programs, the flow charts will help learners contrast the structure of the program text, which is static, with dynamic aspects of program execution, e.g. that sections of the program may not be executed (selection) or may be executed repeatedly (iteration).

### Sketching, walk-throughs, and trace tables

Throughout this unit, simple animations are used to visualise the execution of Python programs, using ‘sketches’ of variables and their values to illustrate how program state (e.g. the values of variables, and the contents of data structures) is modified during program execution. This can provide a visual alternative to trace tables that may be less cumbersome for learners at this stage.

### Subgoal labelling

Subgoals “group a set of solution steps by their purpose” [[source](https://gvu.gatech.edu/research/projects/worked-examples-and-sub-goal-labeling-impacts-learning)]. Subgoal labels are often used in worked examples, to help learners “create a framework for problem-solving” [[source](https://gvu.gatech.edu/research/projects/worked-examples-and-sub-goal-labeling-impacts-learning)]. You can find a short summary about subgoal labels, as well as links to relevant research, in Greg Wilson’s book *Teaching Tech Together*, in the section on [cognitive load](https://teachtogether.tech/#s:architecture-load).

Subgoal labels also relate directly to the Blocks and Relations levels of Schulte’s [Block Model](https://www.researchgate.net/publication/247927531_Block_Model_an_educational_model_of_program_comprehension_as_a_tool_for_a_scholarly_approach_to_teaching), mentioned in the previous lesson. Subgoal labels help learners abstract away from the Atoms level, i.e. individual statements, and understand how blocks of statements form problem-solving units, and how these blocks interact with each other.

Resources are updated regularly - the latest version is available at: [the-cc.io/curriculum](http://the-cc.io/curriculum).



This resource is licensed by the [Raspberry Pi Foundation](https://www.raspberrypi.org/) under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International licence. To view a copy of this license, visit, see [creativecommons.org/licenses/by-nc-sa/4.0/](https://creativecommons.org/licenses/by-nc-sa/4.0/).