Lesson 1: Warm up

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

## Introduction

This introductory lesson serves a double purpose: it reconnects learners with Python, making sure they can read and create simple programs that use selection, and it also takes a step forward, providing a very gentle introduction to lists.

## Learning objectives

* Write programs that display messages, receive keyboard input, and use simple arithmetic expressions in assignment statements
* Use selection (if-elif-else statements) to control the flow of program execution
* Locate and correct common syntax errors
* Create lists and access individual list items

## Key vocabulary

Input, output, variables, assignment, expressions, selection, Boolean/logical expression (condition), list, index, list item

## Preparation

**Subject knowledge:**

* Familiarity with using a Python IDE
* The ability to locate and correct syntax errors in Python programs
* Comfortable use of output, input, and assignment in Python
* Comfortable use of arithmetic and relational operators, as well as arithmetic and logical expressions (conditions)
* Comfortable use of selection in Python programs
* Ability to create lists in Python and access individual elements using their index
* Awareness of common misconceptions that you may encounter in novice learners (see the [relevant section](#_thliz98poo7u) in the pedagogy notes for a list of misconceptions related to this lesson)

**You will need:**

* A Python interpreter and IDE — we suggest using an online environment such as [repl.it](https://repl.it/) or the Mu editor ([codewith.mu](https://codewith.mu/))
* Slides - note that some slides contain animations
* Starter activity
  + Let me check worksheet and solutions
* Activity 1
  + Python code for live coding, starting point
* Activity 2
  + How long until the weekend?worksheet
  + Python code for [task, starting point](https://the-cc.io/py-week-2) (the-cc.io/py-week-2)
* Activity 4
  + As the seasons roll on by worksheet
  + Python code for [task 1, starting point](https://the-cc.io/py-months-1) (the-cc.io/py-months-1)
  + Python code for [task 2, starting point](https://the-cc.io/py-months-2) (the-cc.io/py-months-2)
  + Python code for [task 3, starting point](https://the-cc.io/py-months-3) (the-cc.io/py-months-3)

**You may need:**

* Activity 1
  + Python code for [live coding, end point](https://the-cc.io/py-week-1) (the-cc.io/py-week-1)
* Activity 2
  + Python code for [task, solution](https://the-cc.io/py-week-3) (the-cc.io/py-week-3)
  + Python code for [explorer task, solution](https://the-cc.io/py-week-4) (the-cc.io/py-week-4)
* Activity 3
  + Python code for [demonstration](https://the-cc.io/py-week-5) (the-cc.io/py-week-5)
* Activity 4
  + Python code for [task 1, solution](https://the-cc.io/py-months-4) (the-cc.io/py-months-4)
  + Python code for [task 2, solution](https://the-cc.io/py-months-5) (the-cc.io/py-months-5)
  + Python code for [task 3, solution](https://the-cc.io/py-months-6) (the-cc.io/py-months-6)
* The Python cheat sheets

## Assessment opportunities

Learners can be assessed through their answers in the worksheet for Activity 2. Observation is also very important, including how learners interact through pair programming and collaborate to solve problems.

## At a glance

| **Starter activity**  10 mins | **Let me check**  Use closed-form questions to help learners recall some basic programming concepts in Python and prepare them for the rest of the lesson. |
| --- | --- |
| **Activity 1**  10 mins | **Selection recap**  Use the if-elif-else statement in a live coding activity that checks the current day of the week, in order to refamiliarise learners with the programming environment, using a selection structure. |
| **Activity 2**  10 mins | **How long until the weekend?**  Assign tasks that extend the program from the previous activity, so that learners practise using expressions and selection. |
| **Activity 3**  10 mins | **A number for a name**  Provide a gentle introduction to lists, still within the ‘days of the week’ context. Illustrate how to create lists and access individual items. |
| **Activity 4**  15 mins | **As seasons roll on by**  Assign tasks that mirror the example programs from the previous activity (using months and seasons instead of days), so that learners can practise creating lists and accessing individual items. |
| **Plenary**  2 mins | Ask learners what else they think can be stored in lists, other than day, month, and season names. |

## 

## 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 2–7)  10 mins | **Let me check**  The goal of the Starter activity is to help learners recall some of the basic programming concepts in Python and prepare them for the rest of the lesson. There are four tasks which touch upon output, input, assignments, arithmetic expressions, and selection.  You can use slides 2 to 7 to perform and discuss these tasks interactively with your learners, or you can hand out the ‘Let me check’ worksheet and instruct learners to work on the tasks independently.  We suggest that you use slide 2 and ask learners to think about the first (matching) task as they enter the classroom. When they settle, provide a quick answer and then direct them towards the worksheet for the rest of the tasks.  When providing solutions, use the slides to go over the code in the tasks and make sure that learners can recall the programming constructs involved. This is essential, as the rest of the lesson depends on the learners being able to use them.  Here are some of the things you may want to stress in each task:  **Walkthrough** (task 2)   * Variables and assignment * Using int or float to obtain numerical input * Evaluation of expressions and rules of operator precedence * Code walkthrough   **Walkthrough** (task 3)   * Logical expressions (conditions), branches, and selection * The role of indentation * Code walkthrough   **Identify** (task 4)   * Proper use of terminology |
| --- | --- |
| **Activity 1** (Slides 10–13)  10 mins | **Selection recap**  Group the learners into pairs, as they will be **pair programming** for the rest of the lesson. Remind them of the roles of driver and navigator and explain that they will be switching between these roles as they go through their programming tasks.  Use slide 11 to show learners the syntax and use of the if-elif-else statement in Python.  Then proceed to present the program on slide 12, which reads the current day of the week from the user. Note that the program **uses an integer for each day of the week**, ranging from 0 for Monday to 6 for Sunday (so that lists can be incorporated later on).  Use **live coding** so that, **together with your learners**, you can extend this program to check the day of the week and display whether it’s a weekday or not. Slide 13 shows the code that the live coding session should lead to.  The purpose of this activity is to refamiliarise the learners with the programming environment and provide you with the opportunity to talk them through the thought process of using a selection structure. You also have the chance to highlight any syntactic pitfalls (e.g. colons, indentation) and help learners overcome syntax errors. |
| **Activity 2**  (Slides 14–17)  10 mins | **How long until the weekend?**  Hand out the ‘How long until the weekend?’ worksheet. The tasks involve extending the program from the previous activity and is essentially an opportunity to practise using expressions and selection.  The worksheet contains a syntax checklist. If the learners are faced with syntax errors, remind them that the checklist can help them check for common errors.  Tell the learners to switch between the driver and navigator roles at the start of the activity and at the end of step 3.  Use slides 15 to 17 to provide solutions and brief explanations. |
| **Activity 3**  (Slides 18–29)  10 mins | **A number for a name**  The context of the previous activities involved the days of the week, each assigned a number ranging from 0 for Monday to 6 for Sunday. This activity retains this context and uses it to provide a gentle introduction to lists.  Use slides 18 to 20 to illustrate how a list is created (syntax-wise), what a list is, and what it looks like in memory. Point out to the learners that the line breaks in the definition of a list are unimportant; the items in long lists are often broken down into separate lines to make them more readable.  Use slide 21 to highlight that each item in a list is associated with a numerical index, and then proceed with specific examples of how individual list items can be accessed through their index. The examples start with an integer literal as an index (slides 22–24), demonstrate that an IndexError will occur if the index is out of range (slide 25), and then move on to using a variable (slides 26-27), and an expression (slide 28) as an index.  Go through these examples interactively; ask learners what they think the code will display, before providing the answer. The answers in the final two examples are broken down into simpler steps and animated, to aid learner understanding.  Take every available opportunity to emphasise the fact that list item numbering is zero-based. This is something that learners may take a while to get used to, but the examples (and the corresponding visualisations) are designed to highlight this fine point.  If you like, you can show the code and execute an extended version of the program from the previous activities (the-cc.io/py-week-5), that incorporates the list of days introduced here. |
| **Activity 4**  (Slides 30–33)  15 mins | **As seasons roll on by**  Continuing with **pair programming**, hand out the ‘As seasons roll on by’ worksheet. The first two tasks directly mirror the example programs that were discussed in the previous activity, except the list contains the names of *months*, instead of the days of the week. The third task is mostly about practising the use of multi-branch selection.  The example code from the previous activity is also included at the beginning of the worksheet, as a pair of worked examples. Suggest that learners refer to these examples, should they require help.  Remind the learners to switch between the driver and navigator roles at the start of the activity and at the end of each task.  Use slides 31 to 33 to provide solutions and brief explanations. |
| **Plenary**  (Slide 34)  2 mins | Conclude the lesson by asking learners what else they think can be stored in lists, other than day, month, and season names. Possible answers are numbers, guest names, shopping list items, to-do list items, etc. |
| **Homework** | There will be no homework for this lesson.  However, you may point learners to the [Story time](https://rpf.io/storytime) (rpf.io/storytime) project, in case they are interested in applying what they’ve learnt. This project will also prepare them for some of the ideas they will encounter in the next lessons. |

## Notes on pedagogy

### Common misconceptions

Throughout these lessons, you should be aware of the common misconceptions that may arise with novice programmers. The list of misconceptions below are a selection from Appendix A of Juha Sorva’s Ph.D. thesis, *Visual Program Simulation in Introductory Programming Education*, and are relevant to this lesson. The list will help you avoid 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.

**M23**: Difficulties in understanding the sequentiality of statements.

**M9**: A variable can hold multiple values at a time / ‘remembers’ old values.

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

**M13, 15**: Limited understanding of expressions which lacks the concept of evaluation. Primitive assignment stores equations or unresolved expressions.

**M150**: Difficulties understanding the effect of input function calls on execution.

**M155**: Numbers are just numbers. (Why have int and float separately?)

**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.

**M151**: Confusion between an array and its cell.

### Pair programming

Pair programming is a pedagogical approach which involves learners working together **in a structured manner** to develop programs. The learners assume the roles of driver and navigator, and exchange roles at regular time intervals. The driver controls the mouse and the keyboard to write the code, while the navigator provides guidance and instructions. Since learner pairings are important to the success of this approach, we strongly recommend that you decide on these pairings before the lesson.

You can read more in the [Pair programming Pedagogy Quick Read](https://the-cc/qr03) and the online course[Programming Pedagogy in Secondary Schools: Inspiring Computing Teaching](http://rpf.io/secondarypedagogy).

### Live coding

Greg Wilson, in his book [*Teaching Tech Together*](https://teachtogether.tech/), describes live coding as “the teacher writes code in front of the class while the learners follow along, typing it in and running it as they go”. He calls it “the most effective way to teach programming” and goes on to list the advantages and cite relevant research. Live coding should not be improvised. It is a planned, well-structured “performance”, but it also provides teachers with the opportunity to actively address the unanticipated.

You can find a relevant discussion and a detailed example in the online course [Programming Pedagogy in Secondary Schools: Inspiring Computing Teaching](http://rpf.io/secondarypedagogy).

In many of the lessons here, we suggest using live coding while the learners are paired, i.e. in combination with pair programming.

### Worked examples

### The worksheets handed out to learners throughout this unit often start with a worked example, i.e. an annotated solution to a problem. The tasks that follow will be closely linked to this worked example, so that learners can use it as a reference point. Invariably, these worked examples will also have been presented in class, using live coding, so the reasoning behind the solution will have been fully exposed to the learners.

Worked examples reduce cognitive load and can help learners assimilate new information. You can read more about them in the references at the end of the lesson plan.

### Multiple choice questions and misconceptions

### A lot of the multiple choice questions used within the lessons are targeted towards specific misconceptions, which are outlined in the lesson plan. The same holds for the questions suggested for assessment, with the rationale behind the questions explained in the accompanying solutions.

Exploring the answers to these questions in class in a structured manner (using ‘think, pair, share’ in this case) that involves group and class discussion, links to a teaching method called Peer Instruction. There is evidence to suggest that this method improves retention as well as promoting inclusivity. You can read more about Peer Instruction in the references at the end of the lesson plan, so that you can make the most of these questions and the discussions that follow them.

## Notes

### Python development environment

There is no assumption in this unit that you will be using a particular development environment to write Python programs. There are quite a few alternative options and the best one for you may depend on your situation.

At the time of writing, our suggestion is that you use an online development environment such as [repl.it](https://repl.it), which is also used for all example code in this unit. An alternative online environment is [trinket.io](https://trinket.io/). If you want to use locally-installed software, the [Mu editor](https://codewith.mu/) (codewith.mu) is a friendly, minimal, cross-platform environment, specifically addressed to novice programmers.

### Displaying messages in Python

In these resources, whenever it is necessary to display a message that involves multiple values, we use print with a comma-separated list of values as arguments. For example:

print(lucky, "is my lucky number too", name)

There are a few alternative ways to achieve this (such as string concatenation with the + operator, or using f-strings), but we have made a conscious choice to use this approach in Year 9, as we feel it is syntactically simpler.

### Obtaining keyboard input in Python

In these resources, whenever it is necessary to display a prompt and receive keyboard input, we use print to display the prompt and then input, even though input can be used to display the prompt as well:

print("What’s your name?")

name = input()

Instead of:

name = input("What’s your name?")

We believe this is preferable, as it clearly distinguishes between input and output, and also leads to code that is more easily manageable by learners.

## Additional sources

* [Getting started with Mu](https://projects.raspberrypi.org/en/projects/getting-started-with-mu) project (rpf.io/mu)
* Pedagogy Quick Reads: [Pair programming](https://the-cc/qr03)
* Pedagogy Quick Reads: [Live coding](https://the-cc/qr05)
* The FutureLearn course [Programming Pedagogy in Secondary Schools: Inspiring Computing Teaching](http://rpf.io/secondarypedagogy) contains sections on pair programming and live coding.

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/).