Lesson 1: First steps

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

## Introduction

In this introductory lesson, learners will write and execute their first programs in Python. They will go through the basics of displaying messages, assigning values to variables, and receiving input from the keyboard.

They will familiarise themselves with an entirely different programming environment than the block-based one that they may be accustomed to. It is an environment where they will need to know by heart all of the constructs that they can use, instead of having the options laid out in front of them. It is also an environment in which errors arise if they get a single letter or symbol wrong.

One of the main goals of this lesson (and of the unit) is to support them in this transition, by providing associations with concepts that they are already familiar with and building their confidence in overcoming common obstacles.

Before doing any programming, learners will be introduced to what algorithms and programs are, and how they are different. Through this discussion, they will start to build an understanding of what it means to express instructions in a formal language, and how these instructions can eventually be executed by a machine.

## Learning objectives

* Describe what algorithms and programs are and how they differ
* Recall that a program written in a programming language needs to be translated in order to be executed by a machine
* Write simple Python programs that display messages, assign values to variables, and receive keyboard input
* Locate and correct common syntax errors

## Key vocabulary

Algorithm, program, programming language, program translation and execution, interpreter, programming environment, input, output, variables, assignment

## 

## Preparation

**Subject knowledge:**

* You will need an understanding of the differences between algorithms and programs.
* 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.
* 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
* Activity worksheet
* 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/); if you are using the Mu editor, then you might find the following guide useful to help you prepare: [Getting started with Mu](http://rpf.io/mu) (rpf.io/mu)

**You may need:**

* Printout for producing paper strips for the ‘Following instructions’ activity
* Python code for the ‘First steps’ activity
  + [Task 1 — solution](https://the-cc.io/py-output-1) (the-cc.io/py-output-1)
  + [Task 2 — solution](https://the-cc.io/py-assignment-11) (the-cc.io/py-assignment-11)
  + [Task 3 — solution](https://the-cc.io/py-assignment-12) (the-cc.io/py-assignment-12)
  + [Task 4 — solution](https://the-cc.io/py-input-1) (the-cc.io/py-input-1)
* Python cheat sheets on output, assignment, and input

## Assessment opportunities

You can assess learners’ answers to the ‘First steps’ worksheet. You can also assess learners through observation, for example, by assessing how learners interact through pair programming and collaborate to solve problems.

## At a glance

| **Starter activity**  8 mins | **Following instructions**  To introduce the concept of an algorithm, ask learners to execute “the granddaddy of all algorithms”. |
| --- | --- |
| **Activity 1**  10 mins | **Algorithms, programs, and language**  Outline the main difference between algorithms and programs: language. Explain that executing a program in a high-level language such as Python involves translating it, and prepare learners for their introduction to syntax errors. |
| **Activity 2**  30 mins  6 mins intro  6 mins per task | **First steps in Python**  Familiarise learners with the development environment and use a worksheet to guide them through introductory examples that involve output, assignment, and input. |
| **Activity 3**  2 mins | **Python recap**  To recap the Python content covered in this lesson, ask learners to identify code fragments responsible for output, input, and assignment. |

## 

## 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)  8 mins | **Following instructions**  As learners enter the classroom, show slide 2 to display a sequence of instructions on the board. There is a textual as well as a visual representation of the instructions, so that two different ‘languages’ are used. Ask learners if they think that they can understand and follow these instructions.  **Note:** The algorithm on the slides is historically significant. Even though learners may not be aware of this initially, it is a geometrical version of Euclid’s largest common factor algorithm. Donald Knuth calls it “the granddaddy of all algorithms, because it is the oldest non-trivial algorithm that has survived to the present day”.  Ask for a volunteer to execute the instructions, using 12cm and 18cm paper strips as the starting point (you can use the printout provided to cut strips of any length). Ask them to measure the length of the strips when they have finished; the strips should be 6cm long after two iterations (since the largest common factor of 12 and 18 is 6). Repeat the process with 6cm and 15cm paper strips as the starting point, which should result in two 3cm strips after three iterations (since the LCF of 6 and 15 is 3).  **An algorithm**  Display slide 3. Inform learners that the method that they have just followed was recorded by Euclid in around 300 BC, and it describes a procedure for finding the largest common factor. Introduce the term ‘algorithm’ for describing the method.  Ask learners if they have heard of the term ‘algorithm’ before. They may not be able to provide a concrete definition, but they may have a general sense of what it means, especially in relation to Facebook (an algorithm for creating your feed), Google (a search algorithm), or Amazon (an algorithm for suggesting products). |
| --- | --- |
| **Activity 1** (Slides 6–12)  10 mins | **Algorithms, programs, and language**  Display slide 6, and define what an algorithm is. Stress the importance of language in representing, understanding, and executing algorithms; it would have been impossible to execute the algorithm in the ‘Following instructions’ activity without understanding the language that it was represented in. Point out that some algorithms are only expressed visually, using images, and ask the learners if they can think of some examples (e.g. Lego or IKEA assembly instructions).  Next, provide a definition of what a program is (slide 7). Focus on the main difference between algorithms and programs: language. Programs are algorithms that are expressed following the strict rules of a programming language. This **needs** to be the case because programs will eventually be executed by machines. We cannot write programs in natural language because it is ambiguous and complex. Programming languages have been designed to be unambiguous and simple enough that programs written in these languages can be translated and executed by machines.  **Python interpreter**  Briefly describe the process that a Python program goes through in order for it to be executed. Explain that, in its original form, a Python program is simply text, with no special meaning. Its instructions are not directly executable by our machines. We need to run a program called the Python **interpreter**, which will go over the program, translate it, and have it executed.  If you plan to use an online interpreter, such as Repl.it or Trinket, mention to learners that the Python interpreter does not need to run locally, on your machine.  **Syntax (and errors)**  To prepare your learners for their introduction to syntax errors, briefly discuss syntax rules. Explain that programming languages have rules for syntax, just like natural languages do. However, while humans can sometimes correctly infer meaning when syntax rules are not observed, this is not the case when an interpreter tries to translate and execute a program that violates the syntax rules of a programming language.  Explain that learners will not yet be familiar with syntax errors because it is not possible to make syntax errors in Scratch. However, in text-based programming languages, syntax errors are common and easy to make. Display slide 11 to present a short fragment of Python code and explain that any deviation from uppercase or lowercase, and any missing comma, colon, quotation mark, or bracket, will result in a syntax error and the interpreter will not be able to translate the program. Display slide 12 and tell learners that they should not be discouraged by syntax errors. |
| **Activity 2**  (Slides 13–24)  30 mins  6 mins intro  6 mins per task | **Your first steps in Python**  First, show learners the traditional ‘Hello world’ program and inform them that this is the first program that they will be writing.  **Live coding: Hello world**  Inform learners that they will be doing **pair programming** (find out more in the ‘Notes on pedagogy’ section below). Pair them, briefly explain their roles, and ask them to switch at the end of each task.  Open the development environment that you will be using and give learners a walk-through. Help them locate the features that they will need. Perform **live coding** (find out more in the ‘Notes on pedagogy’ section below): type in the ‘Hello world’ program, making sure that learners follow along with you. Describe the aspects of syntax that they should be careful with: brackets, quotation marks, etc.  Demonstrate how to run the program. Explain that this invokes the interpreter, and show them what they should expect to see as a result. Make a couple of common syntax errors and demonstrate how to read the error messages, then find the line that they refer to and correct the errors.  **First steps**  Hand out the ‘First steps’ worksheet and ask learners to start working on the individual tasks, still using pair programming. When learners have completed a task, ask them to pause and turn their attention to you, so that you can briefly explain some of the finer points in the code for each task (as detailed in the ‘commentary’ slides). These include concepts that they need to be aware of, as highlighted in the slides, i.e. variables, values, and assignment, as well as syntax subtleties.  **Note:** Some of the slides show the Scratch blocks that correspond to (relevant parts of) the Python program, so that learners can make associations with concepts that they are already familiar with. The online course *Scratch to Python: Moving from block-based to text-based programming* includes [a step that draws parallels between Scratch and Python syntax](http://rpf.io/scratchtopython), as well as a relevant cheat sheet at the end. |
| **Activity 3**  (Slide 25)  2 mins | **Python recap**  To recap the Python content covered in this lesson, ask learners to identify code fragments responsible for output, input, and assignment. |
| **Homework** | There is no homework for this lesson. However, if learners are interested in applying what they’ve learnt, you could point them to the [About me](http://rpf.io/about-me) (rpf.io/about-me) project by the Raspberry Pi Foundation. This project will also prepare them for some of the ideas that they will encounter in the next lesson. |

## 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.

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

### Pair programming

Pair programming is a pedagogical approach that 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. You can find out more about pair programming in the resources referenced at the end of the lesson plan.

Since learner pairings are important to the success of this approach, it is strongly recommended that you decide on these pairings before the lesson.

### Live coding

Greg Wilson, in his book [*Teaching Tech Together*](https://teachtogether.tech/), describes live coding as follows: “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 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 out more about live coding in the resources referenced at the end of the lesson plan.

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

The notion of ‘sketching’ variables and their values is referenced in the lesson slides. This links to the idea of using drawings 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.

A very useful tool for step-by-step walk-throughs of Python programs and visualisation of program state is [Python Tutor](http://pythontutor.com/visualize.html) (pythontutor.com/visualize.html).

## Notes

### Python development environment

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

At the time of writing, our suggestion is that you use the [Mu editor](https://codewith.mu/) (codewith.mu), which is a friendly, minimalist, cross-platform environment, specifically designed for novice programmers. There are also advantages to using online coding environments, such as [repl.it](https://repl.it) and [trinket.io](https://trinket.io/).

### 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 other ways to achieve this (such as string concatenation with the + operator, or f-strings), but we have chosen to use this approach in Year 8, as we feel that 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 that this is preferable, as it clearly distinguishes between input and output, and leads to code that is more easily manageable for learners.

## Additional sources

* [Getting started with Mu](https://rpf.io/mu) project (rpf.io/mu) by the Raspberry Pi Foundation
* Pedagogy Quick Read: [Pair programming](https://the-cc/qr03)
* Pedagogy Quick Read: [Live coding](https://the-cc/qr05)

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/).