



Raspberry Pi

Year 8 – Intro to Python programming

Unit introduction

This unit introduces learners to text-based programming with Python. The lessons form a journey that starts with simple programs involving input and output, and gradually moves on through arithmetic operations, randomness, selection, and iteration. Emphasis is placed on tackling common misconceptions and elucidating the mechanics of program execution.

A range of pedagogical tools is employed throughout the unit, with the most prominent being pair programming, live coding, and worked examples.

The Year 7 Programming units ([Scratch Essentials I](#) and [Scratch Essentials II](#)) are a prerequisite for this unit.

Overview of lessons

| Lesson | Brief overview | Learning objectives |
|---------------|---|--|
| 1 First steps | <p>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.</p> <p>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.</p> | <ul style="list-style-type: none"> ● 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 |

| | | |
|---------------------|---|--|
| | <p>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.</p> <p>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.</p> | <p>values to variables, and receive keyboard input</p> <ul style="list-style-type: none"> ● Locate and correct common syntax errors |
| 2 Crunching numbers | <p>In the previous lesson, learners were introduced to displaying messages, assigning values to variables, and receiving input from the keyboard. This lesson will help them gain a deeper understanding of assignments, and explicitly address some of the common misconceptions around the semantics of assignment statements.</p> <p>Learners will also be introduced to using arithmetic expressions and receiving numerical input from the keyboard. These are two key components that will allow them to progress to building more elaborate programs in the lessons to follow.</p> <p>The main activity in this lesson will require learners to construct their own short programs for the first time, through scaffolded tasks.</p> | <ul style="list-style-type: none"> ● Describe the semantics of assignment statements ● Use simple arithmetic expressions in assignment statements to calculate values ● Receive input from the keyboard and convert it to a numerical value |
| 3 At a crossroads | <p>This lesson introduces selection and randomness. These are two features that will allow learners to develop programs with a very diverse range of behaviours.</p> <p>Learners will revisit some of the programs that they have encountered in previous lessons and extend them into more versatile programs that use</p> | <ul style="list-style-type: none"> ● Use relational operators to form logical expressions ● Use binary selection (if, else statements) to control the flow of program execution |

| | | |
|---------------------------|--|---|
| | selection. They will develop a simple number guessing game, which will eventually include randomness. | <ul style="list-style-type: none"> • Generate and use random integers |
| 4 More branches | <p>This lesson progresses to multi-branch selection, then introduces while, the general-purpose iterative structure available in Python.</p> <p>Learners will explore problems that will allow them to deepen their comprehension of when and how selection should be used. For example, they will build programs that check the weather conditions where they are living and display appropriate responses. They will also be introduced to iteration, making sure that they understand the mechanics of how it works, before they go on to build their own iterative programs in the next lesson.</p> <p>At times, learners will import and use functions from ‘home-grown’ modules, i.e. modules that have been created exclusively for the purposes of the lesson. This will give them an insight into how a text-based language can be more powerful than block-based languages, without placing additional cognitive burden on them.</p> | <ul style="list-style-type: none"> • Use multi-branch selection (if, elif, else statements) to control the flow of program execution • Describe how iteration (while statements) controls the flow of program execution |
| 5 Round and round | <p>In the first part of this lesson, learners will be introduced to counting. Counters are important, as they are the simplest example of variables that are used to compute results iteratively, with each new value accumulated over the previous ones.</p> <p>In the second part of the lesson, learners will apply the skills and knowledge that they have developed to create a times tables practice game. It is an example that naturally combines iteration and selection, while also being useful.</p> | <ul style="list-style-type: none"> • Use iteration (while loops) to control the flow of program execution • Use variables as counters in iterative programs |
| 6 Putting it all together | In this final lesson of the unit, learners will apply and consolidate what they’ve learnt by extending the number guessing game that they developed previously into an iterative version that allows them multiple guesses. | <ul style="list-style-type: none"> • Combine iteration and selection to control the flow of program execution |

| | | |
|--|--|--|
| | They will then conclude the unit with a summative assessment quiz. | <ul style="list-style-type: none">• Use Boolean variables as flags |
|--|--|--|

Progression

The fundamental concepts covered in this unit are visually summarised in the concept map for this unit, which can be found in Lesson 1: First steps. The concepts are organised in a layered hierarchy based on [Schulte's Block Model](#), and there is a distinction between static and dynamic aspects of programs (program text vs program execution).

Python syntax and short examples relevant to the contents of this unit are condensed into a set of Python cheat sheets which are part of Lesson 1: First steps.

Please see the learning graph for this unit for more information about progression.

Curriculum links

[National curriculum links](#) (Computing programmes of study: Key Stage 3)

Aims

- can understand and apply the fundamental principles and concepts of computer science, including abstraction, logic, algorithms and data representation
- can analyse problems in computational terms, and have repeated practical experience of writing computer programs in order to solve such problems

Subject content

- use two or more programming languages, at least one of which is textual, to solve a variety of computational problems
- understand several key algorithms that reflect computational thinking; use logical reasoning to compare the utility of alternative algorithms for the same problem
- understand how instructions are stored and executed within a computer system
- design, use and evaluate computational abstractions that model the state and behaviour of real-world problems and physical systems

Assessment

Each lesson includes a set of worksheets that can be used for formative assessment.

Please also see the assessment question and answer documents for this unit. These can be used within the lessons and/or as homework and/or as a question bank for assembling a summative assessment quiz at the end of the unit.

Subject knowledge

At the end of each lesson plan, you will find **lesson-specific notes** providing additional background knowledge, as well as links to external sources.

You may also find the following resources useful for teaching this unit.

- [Computer Science Education: Perspectives on Teaching and Learning in School](#), edited by Sue Sentance, Erik Barendsen, and Carsten Schulte (Bloomsbury, 2018)
- [Teaching Tech Together: How to Create and Deliver Lessons That Work and Build a Teaching Community Around Them](#) by Greg Wilson (Taylor & Francis, 2019).
- [Effective computing pedagogy](#) by the National Centre for Computing Education

Enhance your subject knowledge to teach this unit through the following training opportunities:

Online training courses

- [Programming Pedagogy in Secondary Schools: Inspiring Computing Teaching](#) by the Raspberry Pi Foundation, hosted by FutureLearn
- [Programming 101: An Introduction to Python for Educators](#) by the Raspberry Pi Foundation, hosted by FutureLearn
- [Scratch to Python: Moving from Block- to Text-based Programming](#) by the Raspberry Pi Foundation, hosted by FutureLearn

Development environment

Before delivering this unit, you will need to decide which development environment you will be using. You can use a **local** installation of a Python interpreter and a learner-friendly IDE such as the Mu editor (codewith.mu), or you can use an **online** development environment such as [Repl.it](https://repl.it). You could also use both, but that may be confusing for learners.

Here are some of the things that you should consider before you make your decision:

- **Conceptual understanding:** An interpreter is necessary in order to translate and execute any Python program. Learners need a basic conceptual understanding of what happens when the interpreter is invoked.
 - Local: In a local installation, the interpreter is invoked through the development environment. Learners will probably be able to understand that the interpreter (as well as their Python program) is executed on their computer.
 - Online: Learners access an online development environment through their browser and the interpreter (as well as their Python program) is executed on a remote computer. This may make it more complicated for learners to develop a conceptual understanding of the translation and execution process.
- **Code distribution:** In many cases, you will need to provide Python code to the learners (e.g. as a starting point for a programming activity).
 - Local: You will need to distribute .py files to learners, so consider whether this will be practical in your setting.
 - Online: If you use an online development environment, distributing code is usually as simple as providing a link.
- **Code collection:** It will probably be necessary for learners to submit (some of) their code to you, especially for assessment.
 - Local: You will need to collect .py files from your learners, so consider whether this will be practical in your setting.
 - Online: If you use an online development environment, collecting code will usually involve receiving a link from learners.
- **Module installation:** In some cases, you may need to install additional Python modules.
 - Local: Consider whether it will be practical to install additional Python modules in your setting.
 - Online: If you use an online development environment, a lot of additional modules will be readily available. However, not all Python modules can be used in online environments.
- **Network access:** In Lesson 3, some of the example code provided requires internet access (making use of web APIs to retrieve information).
 - Local: If you are using a local Python installation, you will need to consider (or test) whether this is possible in your setting.
 - Online: This should not be an issue when using an online development environment.

Python code

In the lesson plans, all Python code is provided through Repl.it links that are ready to use. If you need to download a local copy of a program to your machine, append .zip to its URL.

For example, this URL provides access to the last program that learners will develop in this unit:

repl.it/@NCCE/prg4-6-a2-lucky-v6

Appending **.zip** to that URL will allow you to download the program:

repl.it/@NCCE/prg4-6-a2-lucky-v6.zip

Resources are updated regularly — the latest version is available at: ncce.io/tcc.

This resource is licensed under the Open Government Licence, version 3. For more information on this licence, see ncce.io/ogl.