# **Year 9 – Python programming with sequences of data**

## Unit introduction

This unit introduces learners to how data can be represented and processed in sequences, such as lists and strings. The lessons cover a spectrum of operations on sequences of data, that range from accessing an individual element to manipulating the entire sequence. Great care has been taken so that the selection of problems used in the programming tasks are realistic and engaging: learners will process solar system planets, book texts, capital cities, leaked passwords, word dictionaries, ECG data, and more.

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

The Year 7 and 8 Programming units are prerequisites for this unit. It is assumed that learners are already able to write Python programs that display messages, receive keyboard input, use simple arithmetic expressions, and control the flow of program execution through selection and iteration structures.

## Overview of lessons

|  |  |  |
| --- | --- | --- |
| **Lesson** | **Brief overview** | **Learning objectives** |
| 1 Warm up | 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. | * 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
 |
| 2 Playlist | This lesson provides learners with an overview of the operations that are commonly performed on lists: adding, removing, or modifying items; locating or counting occurrences of particular items, etc. Learners are presented with a set of short challenges. They are asked to identify the list operations that would be relevant and apply them to perform the required tasks. Through these challenges, learners will indirectly gain a better understanding of the sort of problems where lists might be useful. They also get accustomed to using dot notation for list methods, although this is not the focus of the lesson.  | * Perform common operations on lists or individual items
 |
| 3 In a while, crocodile | This lesson revolves around iteration using while loops, offering learners a chance to retrieve and apply relevant knowledge. In the first activities, learners will practise using list operations in iterative contexts.Learners will be introduced to the similarities between lists and strings, which will be based on what they already know about operations relating to length, membership, and access to individual characters. The final activity requires them to apply these string operations in an iterative context. | * Use iteration (while statements) to control the flow of program execution
* Perform common operations on lists or individual items
* Perform common operations on strings or individual characters
 |
| 4 The famous for | In this lesson, learners will use a for-loop to iterate over list items. They will initially study a range of examples — to familiarise themselves with its syntax, use, and mechanics — before moving on to apply what they’ve learnt to similar tasks. The activities involve iterating over lists of real-world textual and numerical data, requiring learners to recall and apply knowledge from the previous lessons.The lesson ends with a nod towards using for to iterate over the characters of a string, which may come in handy when learners attempt to solve problems independently.  | * Use iteration (for statements) to iterate over list items
* Perform common operations on lists or strings
 |
| 5 Make a thing | IIn this lesson, learners will be provided with a selection of meaningful mini-projects that will allow them to apply the knowledge and skills they have acquired so far. Each project contains a short introduction that provides context, a detailed description of what learners are expected to develop, and a set of clues that will support them in putting together a solution. Each learner is expected to select one of the mini-projects and complete it within this lesson, or in the first part of the next one. Before starting work on the projects, two short activities will provide learners with additional support around accumulating sums and using for to iterate over strings. This is generally important and will also prove useful in some of the projects. | * Use iteration (for loops) to iterate over lists and strings
* Use variables to keep track of counts and sums
* Combine key programming language features to develop solutions to meaningful problems
 |
| 6 Wrap up | In this final lesson, learners will be given the opportunity to complete their mini-project or explore a second one. They will then take a quiz that will assess their grasp of the programming concepts they have encountered throughout the unit.An optional activity is also provided, for learners that finish early with their assessment quiz, or are simply keen on an additional challenge. |  |

## Progression

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

Python syntax and short examples relevant to the contents of this unit are condensed into a set of Python cheat sheets which can be found in Lesson 1: Warm up.

## Curriculum links

[**National curriculum links**](https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/239067/SECONDARY_national_curriculum_-_Computing.pdf) **(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 how instructions are stored and executed within a computer system
* understand several key algorithms that reflect computational thinking; use logical reasoning to compare the utility of alternative algorithms for the same problem
* 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.

A collection of questions for assessment is also provided that can be used as a summative assessment quiz at the end of the unit.

## Subject knowledge

You will find the following resources generally useful for teaching this unit.

* [*Computer Science Education: Perspectives on Teaching and Learning in School*](https://www.bloomsbury.com/uk/computer-science-education-9781350057111/), 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*](https://teachtogether.tech/), Greg Wilson. Taylor & Francis (2009).
* [Effective computing pedagogy](https://blog.teachcomputing.org/tag/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](https://www.futurelearn.com/courses/secondary-programming-pedagogy) by the Raspberry Pi Foundation, hosted by FutureLearn
* [Programming 101: An Introduction to Python for Educators](https://www.futurelearn.com/courses/programming-101) by the Raspberry Pi Foundation, hosted by FutureLearn
* [Programming 102: Think like a computer scientist](https://www.futurelearn.com/courses/programming-102-think-like-a-computer-scientist) by the Raspberry Pi Foundation, hosted by FutureLearn

## Development environment

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

Here are some of the things you should consider before making 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 grasp 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 to 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 are readily available. However, not all Python modules can be used in online environments.

## 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, you will need to login to repl.it and fork the project to your account. Once you have done this, there is an option to download the project as a .zip file.

Some activities make use of ‘home-grown’ NCCE modules and/or additional data files. These have been signposted in the lesson plans and you will see that the code includes lines such as from **ncce**.textfile import words or from **ncce**.data import dictionary. If you or your learners need to run the programs for these activities **locally**, then you will need to download (and distribute) the code for these projects from the repl.it links provided.

Resources are updated regularly — the latest version is available at: [ncce.io/tcc](http://ncce.io/tcc).

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