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 (ncce.io/year7) are a prerequisite for this unit.

Overview of lessons

Lesson	Brief overview	Learning objectives
1 First steps	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.	• 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

	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.	
2 Crunching numbers	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. 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. The main activity in this lesson will require learners to construct their own short programs for the first time, through scaffolded tasks.	• 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	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	• 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.	• Generate and use random integers
4 More branches	This lesson progresses to multi-branch selection, then introduces while, the general-purpose iterative structure available in Python. 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. 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.	• 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	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. 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.	• 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.	• Combine iteration and selection to control the flow of program execution

	They will then conclude the unit with a summative assessment quiz.	• Use Boolean variables as flags

Progression

This unit focuses on a text-based programming language (Python). It build on the Scratch units learnt in Year 7.

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