# KS4 – Programming

## Unit introduction

This extensive programming unit takes learners from being complete novices to having the confidence to tackle any GCSE-level programming challenge. Essential programming theory is also interleaved into the practical elements of programming to provide tangible links between required knowledge and skills.

The latest pedagogical research has been used to ensure that learners are appropriately scaffolded and challenged as they move through the lessons. Learners that have already programmed with a text-based language might advance through the earlier lessons at a faster rate than planned. Explorer tasks have been provided to help stretch learners that need a further challenge.

## Overview of lessons

|  |  |  |
| --- | --- | --- |
| **Lesson** | **Brief overview** | **Learning objectives** |
| Lesson 1: Translators | In this lesson, learners will be introduced to the notion that humans interpret instructions differently to computers. This is to help them understand that computers need clear and precise instructions in order to perform the expected task. Learners will also be taken on a journey from machine code to high-level languages in order to discover **how** a computer interprets instructions. | * Compare how humans and computers interpret instructions * Explain the differences between high- and low-level programming languages * Describe why translators are necessary * List the differences, benefits, and drawbacks of using a compiler, or an interpreter |
| Lesson 2: Sequence | In this lesson, learners will be introduced to a Python IDE of your choice. They will learn about the function of an IDE and why programmers use these to write programs. Learners will be given some simple code to predict, run, investigate, and modify. Whilst they take their first steps in Python programming, they will also learn about common errors and error types. | * Describe the tools an IDE provides (editors, error diagnostics, runtime environment, translators) * Use subroutines in programs * Define a sequence as instructions performed in order, with each executed in turn * Predict the outcome of a sequence and modify it * Interpret error messages and define error types and identify them in programs (logic, syntax) |
| Lesson 3: Variables | Learners will find out about variables during this lesson. They will learn about the purpose of variables, but also the technical aspects of creating variables to a uniform standard. Variable declaration is not used in Python, so a wider look at this through other programming languages will help learners gain an insight into its meaning. | * Use meaningful identifiers * Determine the need for variables * Distinguish between declaration, initialisation, and assignment of variables * Demonstrate appropriate use of naming conventions * Output data (e.g. print (my\_var)) |
| Lesson 4: Input | During this lesson learners will start to add interactivity to their programs by introducing the input() function. Learners will be given a demonstration of how input() is used in Python and then asked to add this feature in their silly stories that were created last lesson. Learners will then be introduced to the five main data types that they need to understand. This is done first through theory teaching and then a practical activity where learners predict, run and investigate a program that includes the five data types. Whilst learning about input, learners will be introduced to functions and data validation techniques. These will be covered in more detail later on in the course. | * Obtain input from the keyboard in a program * Differentiate between the data types: integer, real, Boolean, character, string * Cast variables by calling a function that will return a new value of the desired data type * Define runtime errors in programs * Define validation checks |
| Lesson 5: Flowcharts | The focus of this lesson is for learners to interpret and create flowcharts. They will use their knowledge of writing simple sequences and subroutines to follow a flowchart, and write the code that it represents. They will be given time to write their own simple flowcharts in order to practise using the symbols that they have learnt during the lesson. This is an introduction to flowchart design and will be built upon throughout the unit. | * Identify flowchart symbols and describe how to use them (start, end, input, output, subroutine) * Translate a flowchart into a program sequence * Design a flowchart for a program |
| Lesson 6: Randomisation | In computer science, random numbers are something that you are likely to use regularly. They are also used in areas such as cryptography, while pseudo-random numbers are used in video games, modelling, and simulations. In this lesson, learners will be introduced to the concept of random numbers using Python documentation. Learners will determine what the random module is capable of, and how random numbers can be generated in Python. | * Be able to locate information using the language documentation * Import modules into your code * Demonstrate how to generate random numbers |
| Lesson 7: Arithmetic expressions | This lesson has been designed to ensure that learners understand the rules of operator precedence when evaluating arithmetic expressions. They will be reminded of BIDMAS, before investigating code that uses various arithmetic expressions. This lesson will prepare them for the next lesson, where they will begin to use conditions in programming. | * Evaluate arithmetic expressions using rules of operator precedence (BIDMAS) * Write and use expressions that use arithmetic operators (add, subtract, multiply, real division, integer division, MOD, to the power) * Assign expressions to variables |
| Lesson 8: Selection | This lesson moves learners on to the next big programming construct: selection. They will be introduced to it initially through a flowchart that demonstrates how a condition can be used to control the flow of execution in a program. They will then learn about definitions for logical expressions and conditions. A short activity has been included to allow learners to grasp how logical expressions evaluate. Next, they will complete a PRIMM activity where they investigate and modify a chatterbot. Finally, peer instruction will be used to assess their learning. | * Define a condition as an expression that can be evaluated to either True or False * Identify flowchart symbols and describe how to use them (decision) * Identify that selection uses conditions to control the flow of execution * Walkthrough code that includes selection (if, elif, else) |
| Lesson 9: Selection challenge | This lesson is an extension of Lesson 8, in which learners completed a PRIMM activity that introduced them to selection. This is the ‘make’ part of PRIMM, where learners will complete a pair programming activity to create a joke machine. This will allow learners to apply their new knowledge to a new, but similar scenario. | * Use selection statements in a program * Identify when selection statements should be used in programs * Write and use expressions that use comparison operators (equal to, not equal to, less than, greater than, less than or equal to, greater than or equal to) |
| Lesson 10: Logical expressions | This lesson will deepen learners’ understanding of logical expressions by introducing the operators AND and OR. It will begin with a Parson's Problem to check the learners’ understanding of selection. It will then move on to an unplugged activity that introduces AND and OR. Learner's will walk through code, and investigate it before writing their own logical expressions. | * Describe how Boolean/logical operators can be used in expressions * Walk through code that use conditions with Boolean/logical operators (AND, OR) * Write and use expressions that use Boolean/logical operators (AND, OR) |
| Lesson 11: Nested selection | This lesson introduces learners to the concept of nesting if statements. Learners will walk through some basic nested statements to check their understanding. They will then follow the PRIMM approach and investigate a ‘guess the animal’ game. Learners will modify the game to improve the functionality of it. | * Define nested selection * Walk through code that uses nested selection * Modify a program that uses nested selection |
| Lesson 12: While loops | This lesson allows learners to find out about using iteration in their programs. It will define iteration, give a code walkthrough of a while loop, and then use a ‘guess the number’ game as a practical example for using iteration. Live coding will be used to turn an if statement into a while loop, and learners will be given a series of challenges to extend the ‘guess the number’ game using programming skills that they have gained during this course. | * Define iteration as a group of instructions that are repeatedly executed * Modify a program to incorporate a while loop |
| Lesson 13: Trace tables | This lesson has been designed to deepen learners’ understanding of a while loop whilst introducing trace tables. Learners will be given an example of how to use a trace table, then they will be given various loops to trace. Trace tables are a valuable part of programming because they allow learners to walk through code and detect errors. | * Use a trace table to walkthrough code that uses a while loop * Use a trace table to detect and correct errors in programs |
| Lesson 14: For loops | This lesson is designed to introduce learners to the concept of a for loop. They will learn about the definition, and be shown how the range function can be used with a for loop. This lesson only focuses on the range function; the wider use of for loops will be covered later on in the unit. This will enable them to be able to compare a while loop and a for loop at an earlier stage in the unit. After a walkthrough of a for loop, learners will be given a times table generator to explore and modify. Finally, learners will compare a while loop to a for loop and develop definitions for each. | * Define a for loop * Walk through code that uses a for loop * Modify a program that uses a for loop * Compare a while loop and a for loop |
| Lesson 15: Data validation | Learners will spend this lesson finding out how to effectively add data validation techniques to their programs. They will learn why they are important, and how to incorporate them. Before this lesson, learners have only used a try and except technique that works once, then breaks the second time. This lesson will show them how to incorporate try and except inside a while loop. It will also show learners how to check the range of a value and if a user input has been left empty. | * Determine the need for validation checks * Use iteration to perform validation checks |
| Lesson 16 and 17: Pseudocode | The next two lessons have been created to introduce learners to pseudocode, whilst giving them the opportunity to design a program. It has been designed over two lessons to give learners the time to get stuck into solving a problem. They will be designing and creating a FizzBuzz quiz as part of the main task.  For the purpose of this unit, we shall use the Teach Computing pseudocode. You may wish to replace this with pseudocode from your chosen examination board. | * Describe the purpose of pseudocode * Translate pseudocode into a program * Design and build a program using pseudocode |
| Lesson 18: Subroutines | Learners need to determine why subroutines are used in programs. Use live coding to improve a calculator-style program by introducing subroutines. Learners will discover the advantages of using subroutines, and how they are used for decomposition. They will also modify a subroutine that uses parameters. | * Describe a subroutine * Describe the purpose of parameters in subroutines * Use procedures that accept arguments through parameters * Describe how subroutines are used for decomposition * List the advantages of subroutines |
| Lesson 19: Functions | During this lesson, learners will be introduced to functions. They have already used built-in functions when programming, but haven’t actually written their own yet. They will learn how to structure a function, and then use a trace table to trace the input, process, and output of various functions. They will then be shown how functions can be used for decomposition, and will take part in a pair programming/live coding activity where they modify a previous program to introduce functions. Finally, homework will be set to test the learners’ understanding of functions. | * Explain the difference between a function and a procedure * Use trace tables to investigate functions * Use functions to return values in programs |
| Lesson 20: Scope | Learners will be introduced to the concept of scope in this lesson. They will be briefed on the definitions, before being given example programs that show how local and global scope work in Python. There is an activity where learners will convert programs that use global variables into programs that pass values through parameters instead. This demonstrates that passing values through parameters can reduce the need for global variables. Learners will then be introduced to constants and how these work in Python, before they complete a lesson quiz. | * Describe scope of variables * Describe how parameters can reduce the need for global variables * Identify when to use global variables * Describe a constant |
| Lesson 21: XOR | Learners were introduced to logic operators in Lesson 10. At that point, they could only use AND and OR. Learners also need an understanding of how the XOR operator works. Python doesn’t have an XOR operator, so they will create their own function for one in this lesson. Learners will be reminded of AND and OR, and then introduced to truth tables. They will learn about XOR, and complete a truth table for the operator. Next, they will dive into designing and creating a function for an XOR operator, using worked examples for support. | * Use a truth table * Describe the function of an XOR operator * Design and create a function for an XOR operator |
| Lesson 22: Structured programming | This lesson introduces learners to the structured approach to programming. They will learn to describe structured programming through a series of exercises. Firstly, learners will be introduced to the concept that all blocks of code should have one entry point and one exit point. They will be shown how to avoid breaks and multiple returns in their programs through some exercises. They will also see how to create a structure chart for a dog walking program. They will complete this in pairs, before attempting to complete the program independently using the structure chart as a guide. Completing these activities will help to build decomposition skills in your learners, whilst they discover the advantages of the structured programming approach. | * Describe the structured approach to programming * Explain the advantages of the structured approach * Use the structured approach in programming |
| Lesson 23 and 24: Create a program | This lesson begins with an exploration of the different ways to test programs, followed by a few example tests to complete. Learners will be introduced to their project, which will be used to check their understanding of all the concepts covered up to this point. As they create their program, they will use their testing skills to perform tests on the program. The program is quite complex, so some scaffolding has been provided to help learners. It is expected that learners will reach different levels of completion for this project. A rubric has been provided to help you assess the project.  Although two lessons have been allocated for this project, you might decide that more time is required. You should use your professional judgement to increase the time allowance if needed. | * Describe iterative testing * Describe the types of testing (erroneous, boundary, normal) * Design and create a program |
| Lesson 25: GUIs | This lesson provides learners with a tour of the world of GUIs (graphical user interfaces). It uses the third-party module guizero to create GUI apps. This will give learners the experience of using a third-party module whilst they explore the differences between event-driven programming and procedural programming. Procedural programs execute code in a sequence, event-driven programs react to events triggered by the user or sensors. Learners will discover how GUIs are different to the sequential programs that they have created so far, before they make their own app that adds two numbers together. They will then be stretched further with the creation of a joke machine. These programs will be familiar to the learners, but will now be represented in a GUI environment. | * Define the term GUI * Import third-party libraries * Use guizero to create an event-driven program that uses a GUI |
| Lesson 26: String handling I | This lesson introduces learners to string-handling techniques. They will be given two new techniques to use (length and access a character), before being given a programming challenge where they will use these techniques in a ‘guess the word’ game. Then, they will be shown how a string can be iterated over, using a for loop. A walk through is used to help learners build a mental model of how this works. They will be asked to use this technique to add extra functionality to their ‘guess the word’ game. Finally, learners will explore the Python documentation for different string methods. | * Describe the function of string operators * Use string-handling techniques * Use for loops with string operations |
| Lesson 27: String handling II | In this lesson, learners are introduced to three new techniques that can be used with string handling. The first two focus on substrings. First, learners are shown how to slice a string and hold it in a variable. Next, learners are shown how to check for a substring within a string. Finally, learners are introduced to ASCII conversion techniques, and are presented with a challenge to create a message decoder.  This lesson covers a number of programming challenges and learners might not complete them all in the time given. Time has been given for learners to spend longer on these challenges in the next lesson, if required. | * Use a substring in a program * Use the in operator to check for a substring * Use chr() and ord() to perform ASCII conversions |
| Lesson 28: String handling III | This lesson allows learners to create a program that uses a variety of string-handling techniques. It also makes use of the random module. Learners have not seen this for a while, so a starter activity has been created to help refamiliarise them with randomisation.  This lesson is *intentionally* shorter than a typical lesson to allow for a continuation of the activities from the last lesson, if required.  The main activity for this lesson is to produce a program that will create a secure password based on three random words. It uses randomisation and string-handling techniques to produce the secure password. | * Create a program that uses string-handling techniques |
| Lesson 29: Arrays and lists | This lesson introduces learners to the data structures: arrays and lists. It defines them, and explains the differences between the two. It then moves on to focus on lists in Python. Learners use lists to create a ‘Simon says…’ game that randomly selects instructions from a list of items. They then move on to learning about append(), and remove(), and perform a shopping list activity where they practise using the methods | * Define a data structure * Define a list and an array * Describe the differences between lists and arrays * Use a list in a program * Append to a list |
| Lesson 30: List methods | This lesson introduces learners to the many other list methods that can be used in programming. These were briefly introduced at the end of the last lesson. It also demonstrates that a list can be traversed using a for loop in much the same way as you can iterate over a string. Learners then complete an activity where they populate a deck of cards and perform some list operations and methods on the deck. Finally, learners are shown that lists can be returned from a function; live coding is used to demonstrate how to do this, and how to create custom-built functions. | * Traverse a list of elements * Use list methods * Create a function that returns a list * Import custom-built functions |
| Lesson 31: Sense HAT I | This lesson uses the Raspberry Pi Sense HAT to explore the use of lists. Learners can either use the physical device or an emulator accessed via Trinket to interact with a Sense HAT. This lesson introduces the Sense HAT with some basic techniques in preparation for the next lesson, where they will be given a programming challenge to complete.  Using a Sense HAT offers a different insight into programming whilst giving learners the opportunity to use a list. A list can be used with the LED matrix to control the individual LEDs.  For this lesson, it might be helpful if learners were aware of how RGB colours work. This is covered in the KS4 – ‘Data representations’ unit. | * Use lists to display output on a physical computing device |
| Lesson 32: Sense HAT II | This lesson gives learners the opportunity to complete three small projects with the Sense HAT. Through the first two activities they will learn how to add random items to a list by using choice() and append(). They will then finish the lesson by creating a Magic 8-Ball that works with the Sense HAT. Finally, learners will have some time to showcase their favourite projects to the class. | * Use randomisation to append items to a list |
| Lesson 33: 2D arrays and lists | Learners are introduced to two-dimensional arrays and lists during this lesson. They will be defined, and then learners will be shown how to access lists and single items in those lists, before they complete an activity to practise these skills. Next, learners will find out how to change and append a 2D list through demonstration, before they create their own password manager programs. | * Define a 2D array and a list * Use a 2D list in a program |
| Lesson 34 and 35: 2D lists challenge | This lesson begins with reminding learners about iterative and final testing. Learners are then introduced to their challenge, which is to create a noughts and crosses game. As they create their program, they will use their testing skills to perform tests on the program. The program is quite complex, so some scaffolding has been provided. It is expected that learners will reach different levels of completion for this project. A rubric has been provided to help you assess the project.  Although two lessons have been allocated for this project, you may decide that more time is required. You should use your professional judgement to increase the time allowance if needed. | * Use a 2D list as part of a programming challenge |
| Lesson 36: Records and dictionaries | This lesson introduces learners to two new data structures. A record and a dictionary. The focus of this lesson is on records and how these can be created and used in Python to form a database. Learners will be shown how to use a dictionary as a record before creating a ‘database’ using dictionaries within a list.  **Note:** This lesson uses a dictionary to represent a record because the record data structure isn’t available in Python. The standard use of dictionaries will be covered in the next lesson. | * Describe the record data structure * Use a dictionary to represent a record in a program * Use a dictionary with a list to represent records in a database |
| Lesson 37: Dictionary challenge | This lesson gives learners the opportunity to use a dictionary data structure in a new context. Learners will create a Caesar cipher encryption program using a dictionary as the cipher wheel. This mini project will allow learners to develop their programming skills through an appropriate challenge and it can also be assessed using a rubric. | * Describe the dictionary data structure * Use a dictionary to produce key-value pairs |
| Lesson 38: Reading text files | Learners will be introduced to text files in this lesson. The focus will be on reading text files, and how the data from a text file can be used within a program. Learners will be stepped through the key methods that are used for reading text files in Python, before they complete two text file challenges. | * Determine the purpose of external data files * Read data from an external text file |
| Lesson 39: Writing to text files | This lesson continues the exploration of text files with the addition of writing and appending to files. Live coding will be used to introduce the two new concepts and mini challenges are used to allow learners to test their understanding of them. | * Write to text files * Append to text files |
| Lesson 40: Work with CSV files | Learners are already familiar with text files and how to read the data from them. A CSV file is still a text file and you can use the same methods and modes that you can with a standard file. However, you might use a CSV file for different purposes and operations. This lesson introduces learners to CSV files and will show them how to read data into a list and a 2D list, before setting them challenges where they will work with the data. | * Describe a CSV file * Read from a CSV file * Use the split() method * Select data from a collection of values |
| Lesson 41: Write to CSV files | Building on from the last three lessons, learners will discover how to write to CSV files. They work with 1D and 2D lists, before converting them to string and writing them to CSV files. There are lots of list challenges this lesson, which should help to enhance their skills in preparation for the final project.  This lesson includes some complex challenges for your learners. You may wish to give them two lessons to complete this work or set one activity for homework. | * Write data from a 1D list to a CSV file * Write data from a 2D list to a CSV file |
| Lesson 42: Being a programmer | This lesson has been designed to round up the learning from this unit. It is the last lesson of new content before they take on their final projects. Learners will discuss the good habits of a programmer before being reminded of why some of the key aspects are good habits. Learners will also hear from industry programmers about their own good practice.  Learners are then shown alternative approaches to programming solutions. This unit has consistently used specific approaches to help them learn different skills. They are now experienced enough to be able to see the other approaches and try them for themselves in their own programs.  Finally learners will be shown how to append to a CSV file, before completing a challenge using their CSV skills and the alternative approaches learnt in today's lesson. | * Determine the good habits of a programmer * Explore alternative approaches to programming solutions * Append to a CSV file |
| Lesson 43: Battle boats | This is the first in a series of eight lessons where learners complete their final programming challenge of the unit. This is their formal assessment for the unit. The project is very challenging, but it does cover everything that they have learnt over this unit. The key is, can they put it all together? This lesson focuses on familiarising learners with the game and the scenario. Learners will write their success criteria for the game to be used next lesson when they design their pseudocode or flowcharts. | * Write success criteria for a challenging project |
| Lesson 44: Battle boats design | This is lesson two of an eight lesson project. Learners will now use their success criteria to help them with their design work. Examples are given to support learners before they spend the majority of the lesson thinking about their designs.  This is a large, challenging project. You may wish to spread this over two lessons or set completion for homework if your learners need more time. | * Design the program for a challenging project using flowchart or pseudocode |
| Lesson 45 to 48: Battle boats code | These four lessons are part of an eight lesson programming challenge. These lessons are designed to give learners time to code their solutions. They will also have the opportunity to complete a code log to track their progress as they go. | * Create the solution for the battle boats program |
| Lesson 49: Battle boats test | This is the penultimate lesson of an eight lesson project. Learners will spend their time completing the final testing of their projects. You might wish to spread this lesson over two lessons if needed, or allow for homework time to finish if appropriate. | * Perform final testing of the solution to a challenging problem |
| Lesson 50: Battle boats evaluate | This is the final lesson of the eight lesson project and the final lesson of the unit. Learners will spend time evaluating their progress against the original scenario given. They should reflect on their strengths and challenges through the completion of their solutions. Finally, learners will submit their projects ready for marking. | * Evaluate a challenging program |

## Progression

This unit progresses learners’ knowledge and understanding of programming.

There are 6 Learning graphs for this unit:

* Learning graph 1 - Sequence
* Learning graph 2 - Selection
* Learning graph 3 - Iteration
* Learning graph 4 - Subroutines
* Learning graph 5 - Strings and lists
* Learning graph 6 - Dictionaries and data files

## Vocabulary

### Vocabulary sheets have been provided for each lesson. These can be given to learners in advance of the lesson to familiarise themselves with the key terms used in the lesson. A full list of the vocabulary definitions can be found in the vocabulary spreadsheet.

## Curriculum links

[**National curriculum links**](https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study)

* Develop their capability, creativity, and knowledge in computer science, digital media, and information technology
* Develop and apply their analytic, problem-solving, design, and computational thinking skills

## Assessment

### Summative assessment

* Battle boats scenario

**Assessment rubric**

* Battle boats rubric

## Subject knowledge

This unit focuses on GCSE-level programming.

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://ncce.io/secondarypedagogy) (ncce.io/secondarypedagogy)
* [Programming 101](https://ncce.io/prog101) (ncce.io/prog101)
* [Programming 102](https://ncce.io/prog102) (ncce.io/prog102)
* [Programming 103](https://ncce.io/prog103) (ncce.io/prog103)
* [Creating an Inclusive Classroom: Approaches to Supporting Learners with SEND in Computing](https://ncce.io/inclusiveclassrooms) (ncce.io/inclusiveclassrooms)
* [Programming with GUIs](https://ncce.io/programmingwithguis) (ncce.io/programmingwithguis)

### Face-to-face courses

* [Python Programming Constructs: Sequencing, Selection, and Iteration](https://ncce.io/f2f-programming) (ncce.io/f2f-programming)

## Development environment

Before delivering this unit, you will need to decide on 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](https://codewith.mu/)), or you can use an **online** development environment such as [repl.it](https://repl.it/). You can, of course, use both, but that may be confusing for your learners.

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

* **Conceptual understanding**: An interpreter is necessary in order to translate and execute any Python program. An elementary conceptual understanding of what happens when the interpreter is invoked is necessary for learners.
  + 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: Consider how convenient it will be to distribute .py files to your learners in your particular circumstances.
  + 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: Consider how convenient it will be to collect .py files from your learners in your particular circumstances.
  + 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 how convenient it will be to install additional Python modules in your particular circumstances.
  + Online: If you use an online development environment, a lot of additional modules are readily available. However, not all Python modules can be used through online environments.
* **Network access**: On occasion, some of the example code provided requires internet access (making use of web API’s to retrieve information).
  + Local: In case of a local Python installation, you will need to consider (or test) if this is possible in your particular circumstances.
  + Online: This should not be an issue when using an online development environment.

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](http://ncce.io/ogl).