From graphical to text-based programming languages

At Key Stage 3 there is a requirement that:

Pupils should be taught to:

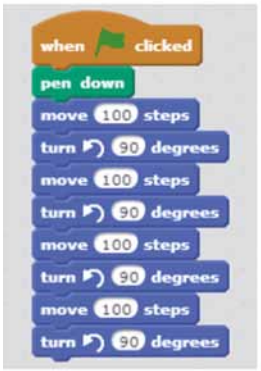
- use two or more programming languages, at least one of which is textual, to solve a variety of computational problems

There is a perception amongst non-specialist teachers that text-based programming is hard. This view is not without foundation if students have to wrestle with the technical details of a new programming language at the same time as they are identifying a problem and developing a program to solve it.

However, utilising pedagogical approaches from other subjects can make the transition from a graphical programming language to a text-based programming language easier to teach and, consequently, easier to learn. Experience has also shown that, once students understand the basics of text-based programming languages, they actively want to use them because they see how limiting graphical programming languages are.

Learning from cross-curricular pedagogical approaches

Teachers, especially at primary level and in modern foreign language teaching, introduce a range of strategies to support the development of literacy skills by students who have varying backgrounds, experiences and levels of linguistic ability. Teachers will use students' existing knowledge of language systems to develop further awareness of how language works. They will also encourage an understanding of the spelling patterns of the language and sound–symbol relationships to support word level awareness. These approaches reinforce and utilise students' existing uses of language and literacy skills. Teachers will facilitate linguistic development in sentence level writing skills where learners demonstrate their word level understanding in meaningful contexts and purposes. Teachers are also aware of the importance of maintaining a balance between improving students' writing sub-skills and enhancing compositional skills (text-level skills) using cohesive and text organisation devices effectively.

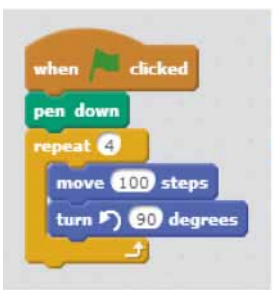



```
from turtle import *
#

fd(100)
lt(90)
fd(100)
lt(90)
fd(100)
lt(90)
fd(100)
lt(90)
```

Drawing a square in Scratch and drawing a square in Python.

We can transfer some of these approaches into the teaching of text-based programming languages. For example, students begin by reading graphical programming constructs and blocks of code, commenting the code to explain what is happening, and then match the graphical programming constructs and blocks of code to their corresponding text-based programming constructs and excerpts of code. This helps them develop an ability to read and comprehend the text-based programming language. Taking a very simple example to illustrate the point:




```
from turtle import *
#

for turn in range(4):
    fd(100)
    lt(90)
```

Drawing a square using the repeat block in Scratch and the 'for' loop in Python.

The graphical blocks of code are slowly removed and students are asked to draw or program those that are missing. When students have developed their confidence in reading the text-based code through this activity, the focus is changed. The graphical blocks of code are reintroduced and the excerpts of text-based code are slowly removed, and students are asked to write or program those that are missing. Leaving some excerpts of text-based code visible provides support for less able students.

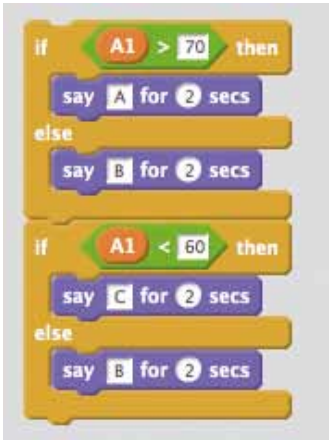
Applying the pedagogy

This same pedagogy can be can be used when students take on new and unfamiliar challenges. For example, students could be presented with the challenge of using a text-based programming language to program a robot that navigates around a maze, and their learning could be scaffolded as follows:

1. Students create a paper net and a paper maze and develop their algorithm unplugged.
2. Students develop and debug a program in Scratch.
3. Students develop and debug a program in a text-based programming language using, for example, RoboMind.

The solution to the problem is reached through several iterations, with students developing their confidence, independence and resilience at each stage.

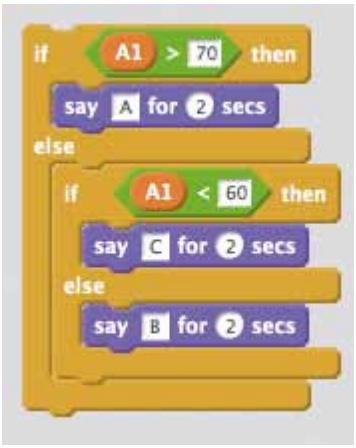
It can even be used to support students' digital literacy skills. Students often struggle to combine two 'if else' statements into a single nested 'if else' statement in a spreadsheet. To help them understand what they need to do, you could model the process in a graphical programming language and then take students' new-found understanding and apply it to a spreadsheet, drawing out the similarities between, for example, Scratch and a spreadsheet formula.



=IF(A1>70, "A", "B")

=IF(A1<60, "C", "B")

Two 'if else' statements in Scratch and Excel.



=IF(A1>70, "A",

IF(A1<60, "C", "B"))

or

=IF(A1>70, "A",
IF(A1<60, "C", "B"))

A single nested 'if else' statement in Scratch and Excel.

Action research

These findings emerged from an action research project and you can find out more about the project by having a look at the following conference paper: Dorling, M. and White, D. 'Scratch: a way to Logo and Python', to be presented at the 46th SIGCSE Technical Symposium in 2015. As the subject of computing grows in confidence in schools, more ideas like this will emerge. Try out some action research yourself, perhaps in partnership with local academics, and see what you can add to the conversation. What can you pinch from other subjects to help you make computing more accessible for your students?